Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Ataques de Negação de Serviço explorando PFS no SSL/TLS: um estudo de viabilidade.

Hugo Gonçalves Marello

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. MSc. João José Costa Gondim

Brasília
2014

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Homero Luiz Piccolo

Banca examinadora composta por:

Prof. MSc. João José Costa Gondim (Orientador) — CIC/UnB
Prof. MSc. Marcos Fagundes Caetano — CIC/UnB
Prof. Dr. Robson de Oliveira Albuquerque — ENE/UnB

CIP — Catalogação Internacional na Publicação

Marello, Hugo Gonçalves.

Ataques de Negação de Serviço explorando PFS no SSL/TLS: um estudo de viabilidade. / Hugo Gonçalves Marello. Brasília : UnB, 2014. 137 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2014.

1. Servidores web, 2. DDoS, 3. segurança, 4. ataques, 5. DoS,
6. SSL/TLS, 7. RSA , 8. diffie & hellman, 9. cifras de segredo perfeito

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Ataques de Negação de Serviço explorando PFS no SSL/TLS: um estudo de viabilidade.

Hugo Gonçalves Marello

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. MSc. João José Costa Gondim (Orientador)
CIC/UnB

Prof. MSc. Marcos Fagundes Caetano
CIC/UnB

Prof. Dr. Robson de Oliveira Albuquerque
ENE/UnB

Prof. Dr. Homero Luiz Piccolo
Coordenador do Bacharelado em Ciência da Computação

Brasília, 7 de julho de 2014

Dedicatória

Gosto de pensar que o ser humano vive de erros e acertos, que cada erro, na verdade, é um passo discreto, um tatear em busca da direção certa. Cada um desses erros é uma possibilidade a menos, um passo mais perto para o caminho certo. E esse documento nada mais é que um pedaço do caminho que tateei, e eu dedico esse pedaço do caminho para todos aqueles que pretendem seguir o caminho e experimentar, ir adiante.

Eu dedico esse documento, esse pedaço de conhecimento, para aqueles que pretendem avançar e cometer menos erros, avançar mais. A humanidade caminhou para o estado atual através da ciência e compartilhamento de informação, tento fazer aqui a minha parte.

Agradecimentos

Agradeço a todos que me ajudaram a passar pela graduação e que me motivaram a seguir em frente mesmo passando por momentos difíceis. Agradeço aos amigos e colegas pelo suporte, as dicas e o companheirismo. Agradeço, também, aos professores mais próximos e ao meu orientador, sem os quais não teria conhecimento, motivação ou condições para escrever esse documento.

Por último, agradeço a Universidade de Brasília (UnB), por tentar ao máximo (na medida do possível) manter um ambiente de ensino de qualidade.

Abstract

A popularização da internet e a inclusão, cada vez maior, da população no mundo virtual transformou seu uso. A internet começou a ter uma importância enorme e começou a ser vista como um bem e uma ferramenta necessária. Entretanto, conforme a internet cresce e se torna incrustada à sociedade, há aqueles que buscam tomar vantagem de seu uso. Surge, então, o crime cibernético, antes um singelo tipo de crime com pouca repercussão agora passa a se espalhar usando da internet e alcançar proporsões desastrosas.

Para suprimir o crime cibernético a área de segurança de redes tomou o papel de fiscal, protetor e divulgador. A segurança de redes tomou o papel de estar sempre atualizada nos mais diversos tipos de ataque cibernéticos e desenvolver ferramentas de defesa sempre atualizadas.

O papel desse documento é divulgar a possibilidade de um tipo novo de ataque usando TLS/SSL(*Transport Layer Security/ Secure Sockets Layer*), mais especificamente um ataque de negação de serviço. O objetivo é mostrar se o ataque é viável e quais medidas de segurança devem ser elaboradas. Para mostrar a viabilidade do ataque foram feitos vários teste usando metodologias diferentes de ataque.

Os códigos fonte das ferramentas de ataque foram omitidos desse documento por motivos de segurança, porém detalhes técnicos e os resultados obtidos são compartilhados, além de análises e sugestões de como elaborar mecanismos de defesa.

Palavras-chave: Servidores web, DDoS, segurança, ataques, DoS, SSL/TLS, RSA , diffie & hellman, cifras de segredo perfeito

Abstract

The popularization of the internet and its major spread among the population, changed its effects and use. The internet started to develop a huge importance and begun to be seen as a tool and a necessity. Otherwise, as the internet grows e become incrusted to society, there are the ones who seek advantage in its use. It's the raise of the cybercrime, once a modest type of crime with low repercussion, now became to spread and reached disastrous consequences.

To overcome the cybercrime, the network security area started to play the role of monitor, defender e knowledge sharer on the internet. Network security worked to be always updated in the newest types of cybercrime and to develop the newest and effectives defense mechanisms.

The main role of this document is to share the possibility of a new type of attack using TLS/SSL(*Transport Layer Security/ Secure Sockets Layer*, more precisely a denial-of-service attack (DoS). The goal is to show the viability of this kind of attack and suggest which defense measures should be taken. To show proof of the effectivity of the attack, lots of tests were made using different kinds of attack methodologies.

The programming codes of the tools used in this document are omitted to prevent harm, nevertheless tecnical details and results from simulations are exposed, with analyses and sugestions on how to elaborate defense mechanisms for the attack.

Keywords: intrusion, exploits, DDoS, security, web servers,DoS, SSL/TLS, RSA , diffie & hellman, perfect foward secrecy

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Durante as últimas décadas registrou-se um grande crescimento na transmissão de dados e informação entre computadores e eletrônicos . Com a popularização da telefonia e a criação da telefonia móvel, o mundo passou a se comunicar cada vez mais. O crescimento de redes de comunicação cabeadas e por ondas de rádio cresceram e tomaram destaque no mundo moderno. Tal ideal de planejar, desenvolver e difundir comunicação de longa distância, inspirou a criação da Internet [18].

A Internet teve vários de seus primeiros protótipos em projetos de comunicação militares, a ARPA (Advanced Research Projects Agency) [1]. Usada nos tempos da guerra fria para comunicação entre zonas militares americanas, a idéia era possuir um canal seguro para informações sigilosas, e ao mesmo tempo um canal resiliente que até mesmo suportasse um possível ataque nuclear. O surgimento da Internet como é conhecida atualmente, entretanto, é um fruto da idéia de expandir o potencial das telecomunicações e difundir informação.

Com a popularização dos computadores a Internet começou a expandir. Usada inicialmente apenas em comunidades acadêmicas, agora cidadãos com acesso a um computador pessoal também poderiam entrar na crescente onda da Internet.

Como um meio de comunicação sob demanda, a Internet se destacou e alcançou rapidamente outros canais de comunicações anteriores, como rádio e televisão. O usuário atual em vez de sintonizar seu aparelho de televisão ou rádio em horários específicos para uma programação específica, agora pode se sentar em frente ao seu computador pessoal e se entreter ou se comunicar com quem quiser e quando quiser. A Internet abria novas e crescentes possibilidades.

Atualmente, a Internet é uma gigantesca rede com milhões de máquinas e usuários conectados [18]. A tecnologia dos equipamentos eletrônicos se superou e começou a integrar o uso da Internet em seus dispositivos, celulares, pda's, consoles de videogames e até mesmo máquinas de lavar agora podem ter acesso à Internet. A Internet avançou em importância e principalmente em comodidade para o cidadão comum, tornou-se um recurso importante. Não um recurso vital à sociedade, mas um recurso necessário ao bem-estar e a comodidade do homem moderno.

Com essa enorme rede presente no cotidiano do homem moderno, consolidou-se novas áreas de estudos, estudos envolvendo redes de computadores e telecomunicações em geral. Essas áreas se dedicam a implantação, planejamento, manutenção, gerência e a segurança dessas redes.

1.1 A segurança de redes

Junto com a evolução da Internet e todas suas comodidades, surgiu, também, o seu mau uso. Ataques de negação de serviço, *malwares*, vírus, *trojans* e vários outros tipos de ataque agora estão dísponíveis na Internet e acontecem diariamente. Cabe a área de segurança de redes estudar medidas para a prevenção e a proteção das redes [10].

A segurança de redes é uma área que cresce sempre associada a outras áreas, pois conforme novas tecnologias começam a aparecer surge a necessidade de protegê-las. Cada vez que novas tecnologias se tornam mais presentes no cotidiano, mais danoso pode ser um ataque usando dessa tecnologia [15].

Grandes empresas possuem servidores e oferecem serviços por meio da Internet. Por exemplo, muitos bancos hoje oferecem inúmeros serviços online e até mesmo efetuam enormes transações comerciais pela rede. Tais serviços estão disponíveis 24 horas por dia ao cliente. Esses serviços são todos hospedados nos servidores dos bancos, e sem a segurança desses servidores, todas as transações de milhares de pessoas poderiam estar vulneráveis junto com suas contas bancárias.

1.2 Objetivo

A ideia desse trabalho de conclusão de curso é avaliar a viabilidade de uma nova forma de ataque de negação de serviço, esse ataque é atual e possivelmente efetivo contra servidores *web* que usam de comunicação segura (TLS/SSL) [9] [8]. Informações sobre ataques de negação de serviço e TLS/SSL serão explicados detalhadamente nos próximos capítulos para compreensão geral.

Até onde se pode averiguar, não há registros de uso de ataques similares ao que se pretende, porém, se viável, o ataque poderá ser usado para incapacitar grandes servidores. O ataque será implementado do zero, baseado em conceitos já existentes. O intuito dessa obra não é disponibilizar um ataque que ponha em risco servidores espalhados pela Internet, mas sim demonstrar a sua existência e alertar a comunidade, deixando claro do ponto da segurança de redes, os seus efeitos.

A prevenção e a proteção, no geral, são facilitadas pelo conhecimento dos efeitos perniciosos que hão de vir. E com a desconstrução do ataque e das vulnerabilidades exploradas pretende-se um maior entendimento do problema, e um ponto de partida para prevenir ou solucionar o problema.

Este documento é constituído de uma primeira parte com um referencial teórico, necessário para a compreensão dos conceitos básicos abordados, seguido por motivações e expectativas. Mais detalhes de implementação, teste e resultados das simulações do ataque serão abordados em capítulos finais.

Capítulo 2

Ataques de Negação de Serviço

O presente capítulo pretende definir os conceitos básicos dos ataques de negação de serviço, além de mostrar exemplos dos ataques, permitindo uma melhor visão do assunto para os próximos capítulos.

A Internet é um conglomerado de redes orientadas ponto a ponto, o que quer dizer que quase todo o processamento da Internet se dá nos nós de origem e destino (clientes e servidores), as sub-redes pelas quais a transmissão ocorrem são focadas na rapidez da transmissão, deixando todo o processamento nos nós [18]. Essa característica, junto com o fato de quase todo tráfego da Internet ser não policiado, permitiu o surgimento dos ataques de negação de serviço.

2.1 Definição

Um ataque de negação de serviço nada mais é que um ataque virtual voltado a interromper o provimento de um serviço específico [21]. Normalmente a intenção do ataque é impossibilitar o acesso de usuários legítimos a uma sub-rede, um computador específico, um servidor ou até mesmo a um serviço disponibilizado online. Podemos citar como alvos mais comuns para esses ataques: servidores de bancos, páginas de compras e vendas, sistemas de cartão de crédito, páginas governamentais e máquinas pessoais [22] [21].

O ataque em si, não permite a extração de informação de seu alvo, diferente de outros ataques não há vazamento de informação apenas a indisponibilidade do serviço atacado. Não se eliminando a possibilidade de ocorrência de ataques diferentes e simultâneos.

Os objetivos dos ataques, quase sempre, são motivados por ambições pessoais onde o objetivo principal é causar dano a vítima. Porém, já foram confirmados vários outros motivos como: prestígio na comunidade *hacker*, atrapalhar empresas concorrentes, chantagem e ataques políticos [21].

O ataque de negação de serviço é popularmente chamado de DoS ou DDoS (no inglês, *Denial of Service* e *Distributed Denial of Service* respectivamente). O DoS consiste em bloquear os recursos da vítima para que usuários legítimos não possam ser capazes de consumi-lo, normalmente, o acesso ou a disponibilidade do recurso são afetados. O DDoS(ataque de negação de serviço distribuído) é uma adaptação do DoS, onde em vez do uso de uma máquina para derrubar o alvo, são usadas várias máquinas infectadas, com o mesmo propósito.

2.1.1 Um breve exemplo

Um simples exemplo de DoS para derrubar servidores *web* é o *Slowloris*. Como descrito na RFC 2616 [23], a requisição de uma página *web* é feita em HTTP, usando TCP. O *Slowloris* é um ataque que tira vantagem das requisições HTTP para esgotar todas as conexões do servidor e deixá-lo ocupado o suficiente para não atender a nenhum usuário legítimo [25].

Uma vez que uma conexão TCP seja estabelecida com o servidor, o atacante manda um pedido parcial de uma página qualquer, porém, esse pedido não está completo, a reação do servidor é esperar pelo resto do pacote que chegará pela conexão TCP. O servidor mantém a conexão ativa enquanto espera pelo resto da requisição, porém o resto da requisição também está incompleto, na verdade todas as continuações estarão incompletas deixando a conexão ocupada e indisponível.

Um servidor pequeno suporta poucas conexões simultâneas e ativas, então com apenas um computador pode-se esgotar todas as conexões de um servidor deixando-o sem serviço. O *Slowloris* apesar de simples já foi bem eficiente em derrubar pequenos e médios servidores e ilustra muito bem a ideia de um DoS que visa esgotar a capacidade do servidor em vez da banda [14] [3].

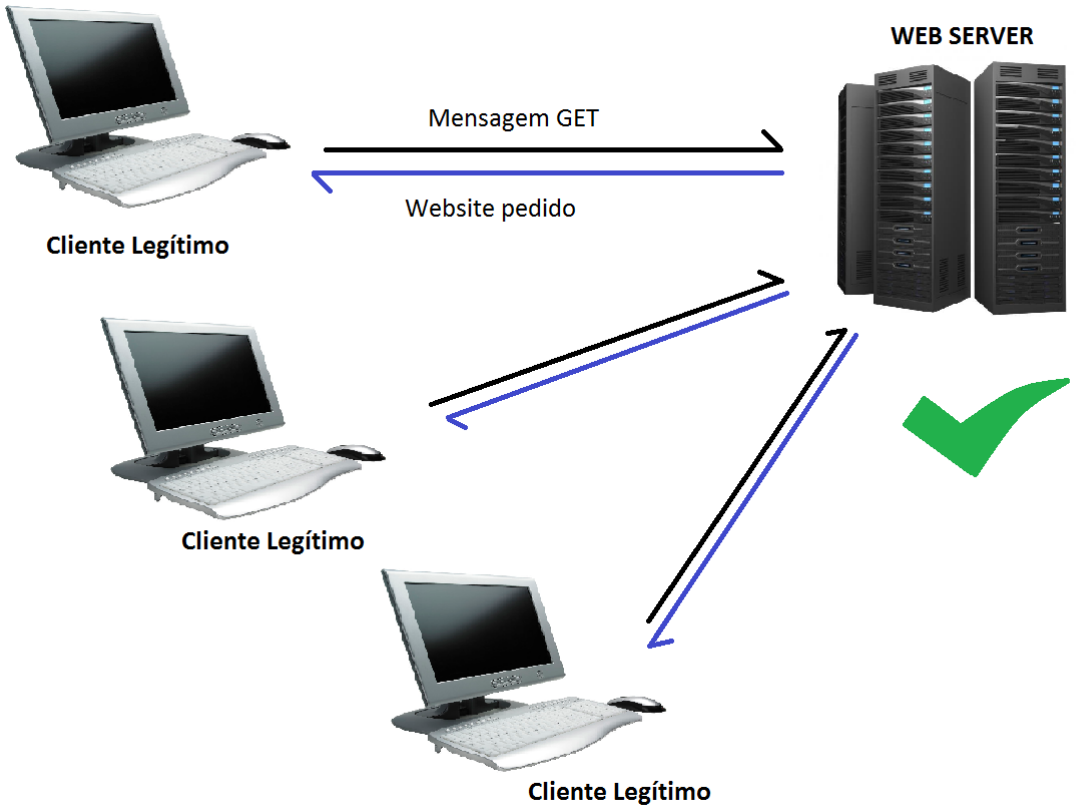

Figura 2.1: Imagem contendo o funcionamento normal de um servidor web, com requisições GET.

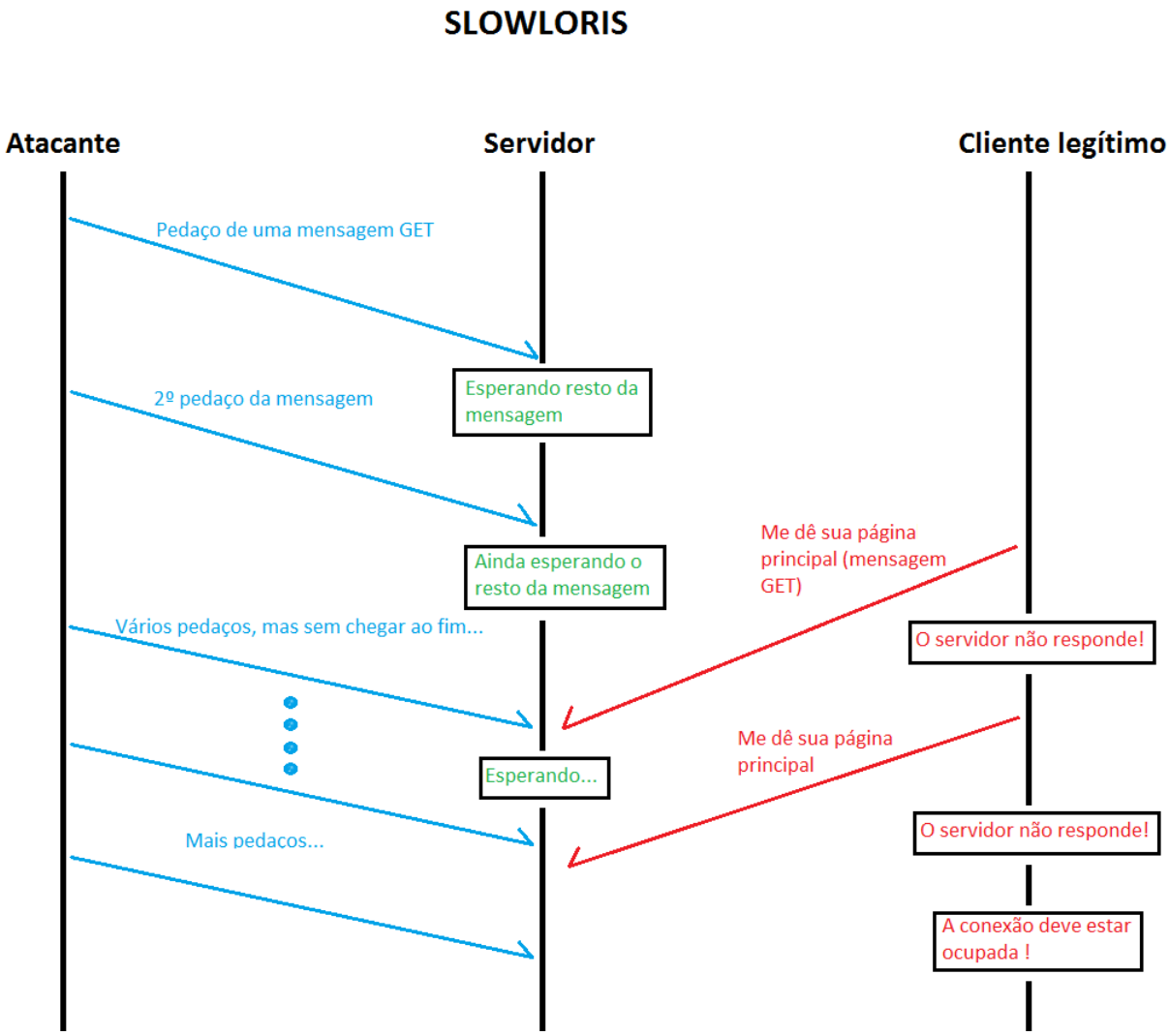

Figura 2.2: Imagem descrevendo o ataque slowloris para uma conexão.

2.2 Dificuldades na segurança contra ataques DoS

Com a constante evolução dos ataques DoS, a segurança e a prevenção contra os ataques se tornam cada vez mais difíceis e exigem constante evolução. Várias ferramentas sofisticadas, poderosas e de fácil uso para executar ataques DoS são facilmente obtidas na Internet, o que aumenta o risco potencial de novos atacantes. Essas ferramentas também são pequenas, simples, de fácil uso e por consequência fáceis de se esconder e evoluir [22]. Os danos diretos causados por um ataque podem incluir: pane do sistema, corrupção de arquivos e indisponibilidade total ou parcial de serviços. Entretanto, danos indiretos podem ser bem graves, alvos de ataques como empresas e orgãos governamentais podem sofrer perdas significativas por não ter um serviço disponibilizado, como perder clientes, deixar de executar um serviço vital a sua sobreviência, perder apoio de clientes, sócios e outras partes envolvidas, perder credibilidade, grandes prejuízos e outras consequências.

Além da constante evolução, os ataques DDoS geram tráfego tão similar e entrelaçado ao de usuários legítimos, que muitas vezes se torna difícil distinguir entre tráfego legítimo e o de um atacante. Muitas das soluções para impedir DDoS's acabam por gastar tanto recurso e banda que acabam por ajudar no ataque, indisponibilizando e consumindo os recursos destinados a usuários legítimos [22].

Existem várias formas de se derrubar um servidor, tendo sido a mais popular por muito tempo, esgotar a banda do alvo. Inundar a banda do alvo com várias sequências de pacotes ininterruptos foi, talvez, a solução mais simples e direta adotada, mas com a evolução do hardware, das redes e dos servidores, a simplicidade se esvaiu. Com a Internet mais adaptada, o número de máquinas necessárias para derrubar um servidor apenas com quantidade se tornou dispendioso, então, outros métodos de ataque surgiram. Aproveitar-se de um protocolo ou uma aplicação mal formulada, causando uma pane ou o desligamento de um alvo, passou a ser uma opção, como citado no exemplo do *slowloris*.

Com a crescente evolução dos DoS e a grande variedade disponível fica difícil o reconhecimento de padrões similares para a detecção dos ataques, sendo que provavelmente muitos mais surgirão. Não há como prever os métodos a serem usados pelos atacantes, porém assim que descobertos, surge a necessidade de uma resposta e uma evolução na segurança. Elaborar uma solução eficiente requer um estudo aprofundado do assunto, o que é a principal idéia desse trabalho de conclusão de curso, demonstrar um ataque atual e viável para dar o pontapé inicial de uma solução criativa antes mesmo que esse ataque se torne popular.

2.2.1 Empecilhos na criação de mecanismos de defesa

O grande impacto e evolução dos DoS levaram a criação de inúmeros mecanismos de defesa (normalmente comerciais), entretanto, o problema continua não solucionado e poucas dessas soluções são adotadas. Uma grande dificuldade da implantação dessas soluções é o fato de que, não adianta apenas a vítima do ataque adotar a solução/mecanismo de defesa, algumas soluções devem ser implantadas em toda a Internet para garantir sua efetividade. Como será visto futuramente, uma rede não segura pode servir de ferrementa de ataque para uma vítima totalmente diferente desconhecida daquela rede.

Grande parte das soluções contra DoS são distribuídas, principalmente pelo fato da vítima não ter muitos recursos disponíveis durante um ataque, já que os mesmos estão sendo bombardeados. Tais soluções precisam ser implantadas ao longo da Internet, evitando máquinas infectadas e redes inseguras, mas não há muita garantia quanto a essas soluções, pois a Internet tem sua gerência descentralizada o que dificulta a adesão geral e acaba por desencorajar a escolha dessas soluções [22].

Uma peculiaridade das soluções distribuídas é que, quem deve adotar a solução, na verdade, não é a vítima do ataque ou alguém da sua subrede, mas sim alguém de fora da situação que está com sua subrede intacta. Tal peculiaridade exige que algum tipo de relação seja estabelecida para que ambos, vítima e portador do mecanismo de defesa, se beneficiem da situação, caso contrário não há motivação ou obrigação para um estranho defender uma outra rede [21].

Outra dificuldade da criação de mecanismos de defesa é a falta de conhecimento, tanto conhecimento sobre o ataque (um dos pontos desse documento) como conhecimento e dados de alguma ocorrência do ataque. Muitas vezes os ataques passam despercebidos e

seus detalhes não são revelados, ou por causarem uma fama ruim à vítima ou por motivos de sigilo (alguns casos governamentais e militares) [22] [21].

Mesmo com tantos empecilhos, várias soluções ainda são implementadas, adotadas e vendidas na Internet. Porém, não há *benchmarks*, comparações ou provas concretas de grande parte dessas soluções, o que também desencoraja a sua adesão. Concertos de pequenas vulnerabilidades em aplicações são fáceis, mas soluções de amplo espectro como as sugeridas são complexas e não há muita garantia de sua eficiência, talvez pela falta de ambiente para se testar ataques, já que por muitos a prática desses ataques mesmo que academicamente é considerada errada.

Capítulo 3

Taxonomia dos Ataques de negação de serviço e mecanismos de defesa

Esse capítulo pretende mostrar os vários tipos de ataques de negação de serviço; não mostrando individualmente os ataques, mas, a sua classificação e a dos mecanismos de defesa atuais contra esses ataques. Este capítulo não é essencial para o entendimento dos demais capítulos, mas é vital para um bom entendimento e aprofundamento nos DoS (ataques de negação de serviço) e consequentemente essencial para a área de segurança, que é o foco principal desse documento.

A segurança, não só de redes, mas em aspecto geral consiste em prevenir, mitigar e solucionar possíveis ameaças. Todos esses pontos têm em comum o fato de necessitarem de conhecimento prévio. Não há como prevenir sem saber o quê há de ser prevenido, não há como diminuir os estragos sem entender o que está acontecendo e quais são os estragos, não há como garantir uma solução imediata a algo que nunca foi visto antes. Conhecimento é sempre o primeiro passo para novas soluções.

3.1 Taxonomia dos DDoS

Um DoS pode ser classificado de várias formas, desde a confecção até o estrago causado. Nessa seção serão abordadas várias classificações dizendo respeito principalmente ao funcionamento do DoS em si [21] [22].

3.1.1 Grau de automação

Como falado anteriormente os DDoS(ataque de negação de serviço distribuído) usam várias máquinas para causar a negação de serviço. Essas máquinas quase sempre são invadidas pelo atacante [22]. A primeira classificação fala sobre isso, o grau de automação dessa infecção. Os primeiros DDoS foram inteiramente manuais, o atacante escaneava por computadores com vulnerabilidades um por um, os infectava usando da vulnerabilidade e instalava o código do ataque manualmente.

Logo depois surgiram ataques semi-automáticos, nesses ataques são implementados hierarquias de mestre e escravo, onde um computador infectado se torna o mestre, infectando e dando ordens para os demais. Nessa classificação a infecção das máquinas e a instalação do código de ataque são automatizadas, porém a ordem de ataque é manual.

O atacante necessita avisar ao mestre o tipo do ataque, a duração, o início e a vítima, só então o mestre começa o ataque dando a ordem a todos os escravos.

Por último, há os ataques completamente automatizados, esses ataques possuem a fase de recrutamento, infecção e instalação do código de ataque automáticas, além das informações todas do ataque já no código de ataque. Desta forma, evitando comunicação do atacante com as máquinas infectadas, a participação do atacante se resume a executar alguns comandos e esperar. Esse tipo de ataque geralmente tem um único uso, já que as informações do alvo e do ataque estão dentro do código. Todavia, esse tipo de ataque deixa vulnerabilidades abertas nas máquinas infectadas facilitando futuros ataques [21].

3.1.2 Comunicação

Em todos os ataques semi-automáticos são necessários algum tipo de comunição entre o atacante e as máquinas infectadas ou entre as máquinas infectadas entre si. E a partir disso podem ser classificados em: comunicação direta e indireta.

Na comunicação direta, as máquinas infectadas precisam se comunicar entre si, e para isso o endereço IP delas é colocado diretamente no código, assim possibilitando a comunicação pela rede. Essa estratégia é vulnerável, pois se uma máquina for revelada obtêm-se o endereço de todas as demais máquinas. Além disso, como a máquina infectada deve possuir uma porta aberta para a comunicação com as outras, é possível achá-la usando escaneadores de rede [21].

A comunicação indireta usa sempre algum tipo de mecanismo de comunicação já existente, algum tempo atrás DDoS usavam canais do IRC [4] para se comunicar e estabelecer os dados do ataque (hoje em dia existem vários outros serviços de comunicação), uma grande vantagem desse método é a anonimidade dada pelo canal, além disso, o fluxo é extremamente difícil de distinguir de um fluxo legítimo de mensagens. Outra vantagem é em caso da descoberta de umas das máquinas infectadas, a informação revelada é escassa e a investigação nesse caso se torna difícil devido a políticas de privacidade e leis internacionais.

3.1.3 Varredura de vítimas

Como já mencionado, os DDoS primeiro precisam recrutar máquinas infectadas e instalar o código de ataque antes de começar o ataque em si. Os DDoS semi-automáticos e automáticos possuem mecanismos automatizados de varredura, essa varredura serve para achar vítimas com vulnerabilidades e infectá-las, popularmente são usados *trojans* e *worms*. Existem inúmeros métodos de se fazer tais varreduras e as mesmas são bem comuns na internet. Análises mostram que milhões de varreduras e *trojans* ocorrem na internet diariamente [34]. Analisar esses padrões de varredura podem revelar um futuro ataque e evitar danos.

As varreduras podem ser aleatórias, por rede local, por lista e por contatos [21]. A varredura aleatória faz com que cada máquina infectada busque por máquinas usando endereços aleatórios, cada uma delas usa uma semente diferente para evitar uma varredura sobre os mesmos endereços. A varredura por rede local tenta buscar todas as máquinas da subrede para então avançar para uma próxima subrede.

A busca por lista é bem diversificada, a idéia principal consiste em uma lista com os endereços a serem varridos. As diversas máquinas infectadas podem assumir posições diferentes na lista usando índices e podem fazer pulos quando encontrarem um endereço já percorrido ou uma máquina já infectada. Além disso podem ser usados diversos procedimentos para evitar colisões na lista e varrer um maior número de endereços. A última varredura usa dos contatos de uma máquina infectada para infectar as demais. Nesse caso, as vítimas são encontradas através de emails, redes sociais e outros métodos de comunicação, *worms* e *trojans* são, então, transmitidos a esses contatos, infectando suas máquinas.

Todas as varreduras citadas ainda podem ser horizontais, verticais, coordenadas e furtivas. As varreduras furtivas são feitas devagar e em parcelas para evitar a sua detecção. As horizontais buscam por apenas uma vulnerabilidade específica em cada máquina, clássica para o uso de *worms*, já as verticais procuram por uma grande quantidade de vulnerabilidades em cada máquina, para só assim avançar para a próxima. Uma varredura coordenada geralmente é usada junto com a varredura por rede local, a mesma tenta combinar as varreduras horizontais e verticais na mesma rede local para aumentar a efetividade da busca.

3.1.4 Vulnerabilidades exploradas pelos DDoS

Os ataques de negação de serviço em si tentam esgotar os recursos da vítima para que ela não possa fornecê-los a usuários legítimos. Os ataques são, então, classificados em duas categorias baseadas em como seus serviços serão negados: volumétricos(inundação e amplificação) e de conteúdo.

Os ataques volumétricos geram uma enorme quantidade de pacotes, que serão enviados à vítima. A vítima fica sobrecarregada e com sua banda saturada, ou no caso de servidores, todo o processamento e(ou) conexões disponíveis ficam sobrecarregados(podendo haver pane ou reinicialização pelo servidor). Um tipo de ataque volumétrico é o de inundação(do inglês *flooding*), esse ataque consiste de um grande número de máquinas infectadas se conectando à vítima e requisitando serviços. As máquinas infectadas costumam requisitar tarefas um pouco mais trabalhosas à vítima, assemelhando-se a usuários legítimos, mas o potencial desse método é a quantidade.

Um outro tipo de ataque volumétrico é o ataque de amplificação [22]. Esse tipo de ataque consiste em gerar uma enorme quantidade de pacotes para a vítima, mas necessitando de menos máquinas. O ataque consiste de usar algum tipo de ferramenta ou vulnerabilidade para amplificar a quantidade de tráfego gerado e direcioná-lo à vítima. Um bom exemplo desse tipo de ataque é o *smurf* [6]. O ataque *smurf* consiste em enviar vários pings(requisições ICMP que necessitam de respostas) para uma subrede em seu endereço de broadcast, porém esses pings possuem o seu endereço de remetente forjado(*ip spoofing*[1]), todas as máquinas da subrede respondem esse ping para o endereço forjado, a vítima. Um único pacote enviado gera vários pacotes de resposta direcionados a vítima, amplificando o dano que cada máquina atacante causaria à vítima.

O problema do ataque *smurf* foi mitigado evitando-se que redes pela Internet funcionassem como amplificadores [25], para isso desabilita-se a propagação de pacotes de

[1] O *ip spoofing* é uma técnica usado por atacantes que consiste em mudar o campo de remetente dos pacotes IP por um outro qualquer, dependendo da finalidade.

controle (*pings*) em *broadcast* nos nós da subrede. Um outro ataque volumétrico de amplificação mais atual consiste de usar servidores de DNS(*Domain Name System* [2]) para amplificar a quantidade de tráfego e direcioná-lo [27] assim como o *smurf*. A figura 3.1 a seguir exemplifica melhor os ataques de amplificação.

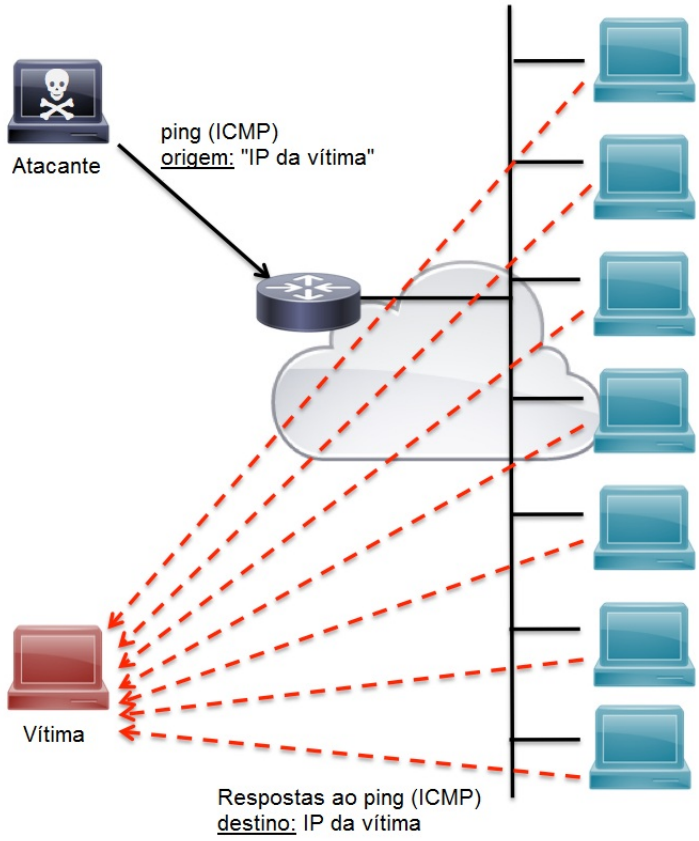

Figura 3.1: Imagem descrevendo o funcionamento de um ataque *smurf* [25](adaptada para o português).

Esses ataques ressaltam que mesmo uma rede protegida pode se tornar insegura, pois outra rede não segura pode amplificar ou causar grandes problemas. A segurança de redes, nesse caso, tem um enorme empecilho, a falta de segurança de alguns pode comprometer a segurança de todos, e a internet é um vasto emaranhado onde nem todos estão bem protegidos.

Os DDoS também podem ser de conteúdo(também chamado vulnerabilidade), esses ataques, no geral, enviam um número alto de pacotes mas não são caracterizados por isso. Essa categoria de ataque explora de falhas de implementação, características ruins e mau uso de protocolos e aplicações pela vítima para consumir seus recursos ou causar uma pane. Como citado no capítulo anterior, o *slowloris* é um ataque de conteúdo que usa de

[2]DNS é um serviço usado na internet inteira onde enderços com letras são convertidos para seu correspondente endereço IP. Um endereço hipotético *www.hipotese.com* seria subistituído por um endereço IP hipotético *189.100.100.100* .

uma vulnerabilidade em como um servidor web trata requisiões HTTP para ocupar todas as suas conexões.

Um outro exemplo bem popular de ataque de conteúdo é o TCP SYN, esse ataque é bem parecido porém em vez de explorar falhas no tratamento de aplicações(no caso HTTP) ele explora falhas em cima do protocolo TCP. No protocolo TCP, para o cliente se conectar ao servidor ele necessita mandar um pacote de início, receber um pacote de início do servidor e então confirmar, três pacotes no total (*handshake*). O ataque consiste em enviar o primeiro pacote, receber o pacote do servidor e nunca mandar a confirmação. Fazendo isso para todas as conexões do servidor, ele ficará com todas elas ocupadas esperando confirmações que nunca chegaram.

O objetivo desse trabalho de conclusão de curso, como já mencionado, inclue a implementação de um desses DDoS de conteúdo voltado totalmente para a área didática, com o único intuito de ajudar a todos os interessados a criar novos mecanismos de defesa para que esse ataque possa ser evitado antes mesmo de se tornar uma ameaça de verdade.

3.1.5 Dinâmicas do fluxo de ataque

Durante os ataques, cada máquina infectada manda fluxos de dados à vítima, direta ou indiretamente. Esse fluxo de dados pode ser constante ou variar com o tempo, dependendo da intenção do atacante. A grande maioria dos ataques costuma mandar um fluxo constante de dados e sempre o máximo possível, para garantir uma negação de serviço completa e mais eficiente. Um fluxo constante de ataque, entretanto, revela facilmente o ataque [22].

Ataques com fluxo variável são realizados para evitar ou atrasar sua detecção. Alguns ataques costumam aumentar gradualmente o seu fluxo para ir degenerando a vítima aos poucos, evitando detecção. Outros ataques involvem fluxos intermitentes sincronizados entre todas as máquinas infectadas, tal estratégia permite períodos espaçados de negação de serviço, que podem se adaptar a reação da vítima e a mecanismos de defesa. Ataques com fluxo variável também podem dividir todas as máquinas infectadas em subgrupos, esses subgrupos ficam períodos intermitentes ativos ou inativos, assim deixando bem difícil a detecção por parte da vítima [21].

3.1.6 Tipo de vítima e impactos

Uma vítima de DDoS pode sofrer o ataque de diferentes formas, o dano pode ocorrer devido a uma vulnerabilidade em alguma das camadas do servidor ou em sua própria infraestrutura. Dependendo do tipo de dano causado e qual parte da vítima foi afetada o ataque pode ser caracterizado de maneiras diferentes. [21].

Vítimas podem ser classificadas por sofrerem problemas em suas aplicações, podendo ter todas as suas outras funcionalidades ativas, mas tendo a aplicação interrompida ou ocupada indeterminadamente, a maioria dos ataques que afetam aplicações usam de vulnerabilidades e geralmente possuem tráfego bem similar a usuários legítimos. Um outro tipo de ataque afeta o *host* diretamente , afetando a comunicação de usuários legítimos, ocupando todas as conexões, interrompendo seus mecanismos de comunicação ou fazendo a vítima ter falhas críticas e reiniciar [21]. Um exemplo desse ataque é o TCP SYN citado em subseções anteriores [22].

Existem, também, aqueles ataques contra recursos da vítima, geralmente visando recursos vitais. Entre eles podemos destacar servidores de DNS específicos, roteadores, e nós com gargalos. Esses ataques podem ser evitados usando uma topologia de rede robusta e aumentando a quantidade de recursos críticos. Como exemplo, ataques voltados diretamente à redes são caracterizados por consumir toda a banda disponível do alvo para que a disposição física do canal não possa permitir o envio de pacotes legítimos.

Por último, o ataque de infraestrutura que visa indisponibilizar serviços essenciais ao funcionamento da internet e consequentemente à vítima. Entre as infraestruturas alvo desses ataques encontram-se servidores de DNS, roteadores de larga escala, protocolos de roteamento e servidores de certificados. Ataques dessa intensidade são devastadores e só podem ser resolvidos juntando-se ações coordenadas de múltiplos agentes da internet.

Além dessa categoria de parte afetada da vítima, podemos caracterizar ataques no grau de impacto causado. Nessa categoria encontram-se ataques disruptivos e degradantes. O objetivo dos ataques degradantes é consumir parcialmente os recursos da vítima, diferentemente de um ataque disruptivo, esse ataque pode permanecer indetectado por um bom tempo. Mesmo parecendo um ataque mais fraco, os ataques degradantes podem ocasionar lentidão e inacessibilidade à alguns consumidores legítimos. No caso de empresas, pode chegar a influenciar a escolha de um cliente sobre um possível concorrente ou até mesmo causar uma queda no lucro devido a alguns comsumidores não conseguirem conexão com a empresa.

Os ataques disruptivos afetam totalmente os recursos da vítima, os tornando totalmente inacessíveis a úsuarios legítimos. Grande maioria absoluta dos ataques reportados até hoje são dessa categoria [21]. Dentre esses ataques que minam todos os recursos da vítima podemos citar ataques que só afetam a vítima no período ativo do ataque (na qual a vítima se recupera totalmente assim que o ataque cessa), ataques sem recuperação (onde há necessidade de reparação de *hardware*) e ataques que necessitam de intervenção humana (reiniciar, reconfigurar e etc). Os ataques disruptivos sem recuperação são teoricamente possíveis porém nunca houveram provas tangíveis de seu uso [21].

3.2 Classificação dos mecanismos de defesa

Os mecanismos de defesa contra DDoS e DoS atuais podem ser classificados baseados em sua reação ao ataque, podendo ser preventivos e reativos [22]. Os mecanismos preventivos são baseados em uma reação pré-ataque, a idéia geral desses mecanismos é evitar que os ataques comecem, assim evitando o ataque por completo ou garantindo que a vítima consiga suportar o ataque e mantenha a disponibilidade dos seu serviços. Por outro lado, os mecanismos reativos tentam aliviar o impacto de ataques à vítima. Eles se concentram em detectar e reagir ao ataque , assim mitigando as consequências do ataque. Um bom mecanismo reativo tem que se preocupar com uma boa detecção, evitando falsos positivos e uma boa reação garantindo a disponibilidade dos serviços oferecidos.

3.2.1 Mecanismos de defesa preventivos

Defesas preventivas modificam sistemas e protocolos da internet para eliminar a tentativa de possíveis ataques, entretanto, nenhum mecanismo de defesa preventivo é totalmente eficiente, pois a implantação global desses recursos não pode ser garantida. Tal

desvantagem ainda não impede a implantação desses mecanismos, pois mesmo não sendo totalmente garantido, a frequência e o impacto dos ataques costuma diminuir significativamente.

Um dos tipos de defesa preventiva modifica sistemas em si para impedir que violações desses sistemas desencadeiem um DDoS. Essa defesa tem como principal objetivo impedir que atacantes usem de bugs e falhas de segurança para infectar sistemas. Uma infecção em um sistema pode obrigá-lo a se tornar um novo atacante que será usado em um DDoS, muitas vezes até passando despercebido. O mérito desse tipo de defesa está em impedir que um atacante infecte máquinas, assim perdendo seu 'exército'. Entre os exemplos dessa defesa se encontram sistemas de monitoramento de máquinas, aplicações que possem atualizações periódicas de segurança, firewalls, sistemas de prevenção de intrusões e defesas contra worms.

Um outro tipo de defesa preventiva é a segurança de protocolos. Alguns ataques costumam explorar o mau funcionamento e implementação de protocolos para exaurir os recursos da vítima, assim como o TCP SYN e o slowloris. Um bom exemplo é o *ipspoofing*, como os protocolos de roteamento de pacotes não fazem validação de endereços, atacantes forjam os endereços de origem dos pacotes para não serem descobertos ou causarem ataques, uma melhor implementação com validação de endereços evitaria tal problema. Geralmente, defesas preventivas na segurança de protocolos consistem de adaptações nos protocolos e publicações de *designs* de implementações seguras.

Um DDoS pode ser prevenido consertando-se falhas de segurança em sistemas e protocolos, mas também pode ser evitado garantindo que serviços estejam semprem disponíveis, inclusive durante um ataque. Um servidor pode muito bem adquirir maior abundância de recursos garantindo maior disponibilidade para clientes e exigindo uma quantidade de atacantes muito maior para um ataque efetivo. Todavia, essa solução envolve um custo monetário elevado, mesmo se provando eficiente [21]. Outra maneira de garantir recursos é usar políticas de divisão de recursos e justiça, usar políticas de escalonamento de recursos garante que todos os usuários inclusive não legítimos tenham acesso ao recurso(mesmo que limitado durante um ataque).

3.2.2 Mecanismos de defesa reativos

Os mecanismos de defesa reativos entram em destaque na hora do ataque em si. Esses mecanismos detectam um ataque em andamento e usam de diversas estratégias para aliviar o dano causado pelo ataque, deixando a vítima operante. Nessa categoria de mecanismos, uma detecção boa(baixos falso positivos) e rápida é fundamental, pois gera uma janela maior de resposta.

A primeira etapa de uma defesa reativa é detectar o ataque, isso pode ser feito através de detecção de padrões, detecção de anomalias e mecanismos externos de detecção. A detecção por padrões já vem fazendo sucesso na história dos antivírus, ela consiste de um banco de dados com todas os padrões de ataques conhecidos e o monitoramento de todas as comunicações para detecção desses padrões. A grande vantagem desse mecanismo de detecção é a facilidade da detecção e a ausência de falsos positivos [21], no entanto, pequenas variações de ataques passam despercebidos e novos ataques não constumam constar no banco de dados. Esse mecanismo de detecção exige constantes atualizações para evitar esse casos.

Outro método de detecção é por anomalias na rede, esse mecanismo costuma manter modelos de funcionamento do sistema em condições normais, como condições do tráfego na rede e da performace do sistema. Esses modelos feitos em momentos rotineiros guardam padrões de como o sistema funciona livre de ataques, o sistema, então, é comparado periódicamente com esses modelos para a detecção de uma possível anomalia. Esse método permite a detecção de novos ataques mas possue alguns casos de falsos positivos.

A detecção por anomalia faz a comparação do sistema com os modelos rotineiros e então toma um limiar em consideração, pois um leve desvio dos modelos é plausível e não indica necessariamente um ataque. A escolha desse limiar pode influenciar em maior número de falsos positivos ou alguns ataques passarem despercebidos. Uma outra decisão da detecção por anomalias é como o modelo de comparação será criado. Esse modelo pode ser estático, gerado em momentos escolhidos e focados na corretude dos protocolos, ou evolutivo, sendo modificado e aprendendo conforme o comportamento do sistema evolue [21]. Esses modelos estáticos são vulneráveis a ataques sofisticados que abusam corretamente dos protocolos, já os evolutivos são atualizados constantemente, sendo alvos de ataques com aumento gradual de fluxo, nos quais a criação de novos modelos é alterada e a detecção falha.

A última maneira de detecção é o uso de mecanismos externos, nesse caso a detecção não é realizada pela vítima e sim por outros agentes, o agente detecta o ataque e notifica a vítima com detalhes do ataque.

3.2.3 Estratégias de resposta ao ataque

Responder a um ataque de DDoS é necessário devido a falhas nas estratégias preventivas e quando defesas reativas já detectaram o ataque. A idéia de responder ao ataque é evitar a negação de serviço, aliviando o impacto do ataque e garantindo que usuários legítimos continuem usando dos recursos com danos mínimos.

Uma das possíveis reações esperadas é achar o atacante ou quais máquinas estão sendo usadas durante o ataque, para assim adotar novas ações. Geralmente são usados mecanismos de *traceback* para isso[3].

Quando um fluxo de pacotes é suspeito mas não está bem caracterizado como um ataque, uma das soluções é limitar a banda. Limitar a banda de certos pacotes diminue o impacto geral na vítima, e em caso de falsos positivos ainda deixa o usuário legítimo ter acesso a seus recursos, mesmo que limitado. A desvantagem desse método está em permitir que algum fluxo malicioso passe, dessa forma ataques de larga escala ainda podem ser efetivos.

Uma outra opção é a filtragem. Ao se detectar um fluxo de dados malicioso pode-se filtrar totalmente o fluxo mal-intencionado, essa solução evita completamente alguns tipos de ataque, mas em caso de falsos positivos usuários legítimos sofrem negação de serviço pelo próprio sistema.

A última estratégia aqui apresentada é mais drástica e consiste de reconfigurar a rede da vítima ou de uma rede intermediária. Em caso de ataques pode-se alterar a topologia da rede (mudando a configuração de nós, tabelas de roteamentos e etc..) para se isolar o

[3]Como o protocolo IP não exige autenticação de origem dos pacotes (como no *ipspoofing*), mecanismos de *traceback* usam de outros métodos para descobrir o endereço real de um pacote ou um fluxo de pacotes.

atacante. Essa solução é dispendiosa e também pode auto-infligir negação de serviço no caso de falsos positivos [21].

3.2.4 Implantação do mecanismo de defesa

Uma última consideração a ser ponderada é o local da implantação do sistema de defesa. Grande maioria das soluções são sempre colocadas na rede da vítima, pois a mesma é quem sofre o ataque e por isso é a mais motivada em ter uma boa segurança e arcar com seus custos.

Outro bom local, mas meio raro, é o uso de uma rede intermediária. Essa rede, que geralmente não está sobrecarregada, se torna a responsável por defender a rede vítima do ataque, o que implica na necessidade de alguma relação ou cooperação entre as duas redes. Esse tipo de defesa é raro e muitas vezes só se torna eficiente com alta adesão na internet, o que não pode ser garantido.

A rede de origem do ataque também é um lugar viável de implantação, principalmente para defesas preventidas. O objetivo da implantação nesse local é evitar que uma máquina local seja infectada ou tente infectar outras, assim evitando a criação de um 'exército' e consequentemente, evitando o ataque. Essa solução entretanto é pouco usada pois necessita de alta adesão na internet e há pouca motivação para sua adoção.

Capítulo 4

TLS/SSL

Algumas informações sigilosas não podem correr livremente na internet, há um alto risco de interceptação, existe a chance de alguém se passar pelo destinatário e ainda há a chance da mensagem ser alterada em trânsito. Para resolver todos esses problemas e permitir um fluxo de informações privadas e autênticas usa-se de criptografia e do protocolo TLS (*Transport Layer Security*).

O protocolo TLS é sucessor do SSL(*Secure Sockets Layer*) e serve justamente para garantir integridade, sigilo e autenticidade de pacotes enviados pela internet [9] [8]. O protocolo não está definido como uma camada da pilha TCP/IP ou do modelo OSI, mas age como uma camada abstrata entre a camada de aplicação e a camada de transporte [32]. O protocolo funciona em cima de fluxos TCP, pois há garantia de conexão entre as duas partes.

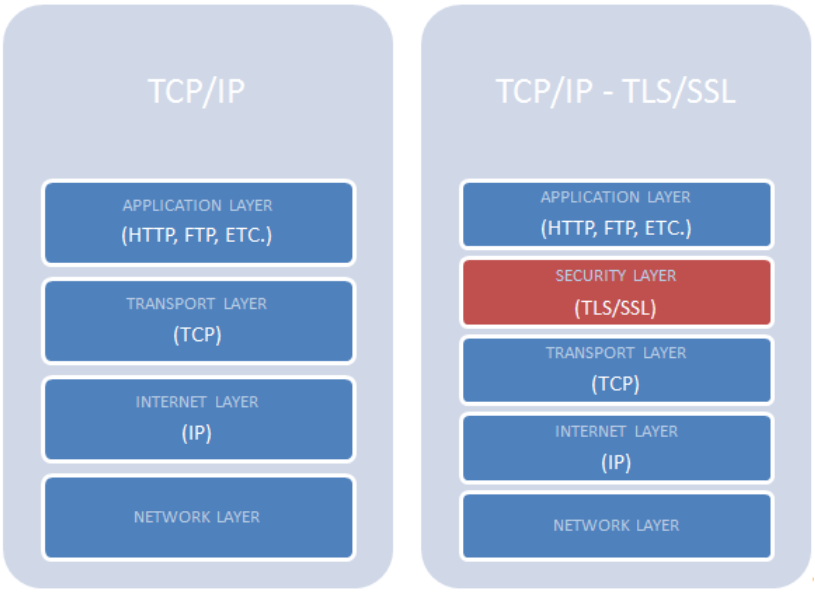

Figura 4.1: Esquema de localização do protocolo TLS/SSL na pilha TCP/IP [32].

O TLS e o SSL são protocolos criptográficos que usam criptografia assimétrica e simétrica, além de certificados, para garantir uma conexão segura sem extravio de informação [9] [8]. O funcionamento do protocolo TLS/SSL diz que primeiramente o cliente

deve requisitar ao servidor desejado uma conexão segura TLS/SSL, mas alguns servidores costumam induzir o cliente a tal conexão segura. O protocolo começa com o cliente requisitando a conexão segura através de um *handshake*, que seria um acordo entre as partes de como a conexão irá acontecer. É durante o *handshake* que ocorrem as verificações dos certificados para garantir que ambas as partes são quem dizem ser.

A conexão funciona em cima de uma outra conexão TCP, e depois de feito o *handshake* TLS/SSL todos os dados transmitidos entre as duas partes serão cifrados de tal forma que apenas as duas partes conheçam o conteúdo transmitido, até que a conexão seja encerrada.

4.1 Criptografia simétrica e assimétrica

Para entender o funcionamento do protocolo TLS é necessário um conhecimento prévio de criptografia, pois o mesmo usa amplamente de criptografia simétrica e assimétrica.

Criptografia assimétrica ou criptografia de chave pública é um tipo de criptografia onde é usado um par de chaves (diferentes entre si) para cifrar e decifrar uma mensagem, para que a mesma seja transmitida de forma segura através de um canal inseguro. Inicialmente uma organização que queira usar criptografia assimétrica pede um par de chaves a uma autoridade certificadora, essa autoridade irá guardar a chave pública da organização e toda vez que um novo usuário quiser se comunicar, a autoridade certificadora irá confirmar que tal chave é de fato a chave pública correta da organização.

A chave pública serve para cifrar a mensagem, que só poderá ser decifrada por quem possuir a chave privada (a organização). Por mais que todos os usuários conhecam a chave pública, apenas a organização sabe decifrar, logo, usuários usando de criptografia assimétrica podem ter certeza que estão se comunicando verdadeiramente com a organização desejada (garantido pela autoridade certificadora).

Em algoritmos cujo o processo de cifragem e decifragem coincidem, como o RSA, as mensagens também podem ser cifradas com a chave privada e decifradas com chave pública (processo contrário). Esse processo permite que a identidade do usuário também possa ser autenticada quando usado corretamente [30].

A criptografia simétrica é mais simples, nela é usado apenas uma chave para cifrar e decifrar mensagens. No entanto, ambas as partes devem conhecer a chave préviamente, que deve ser transmitida através de um canal seguro [30].

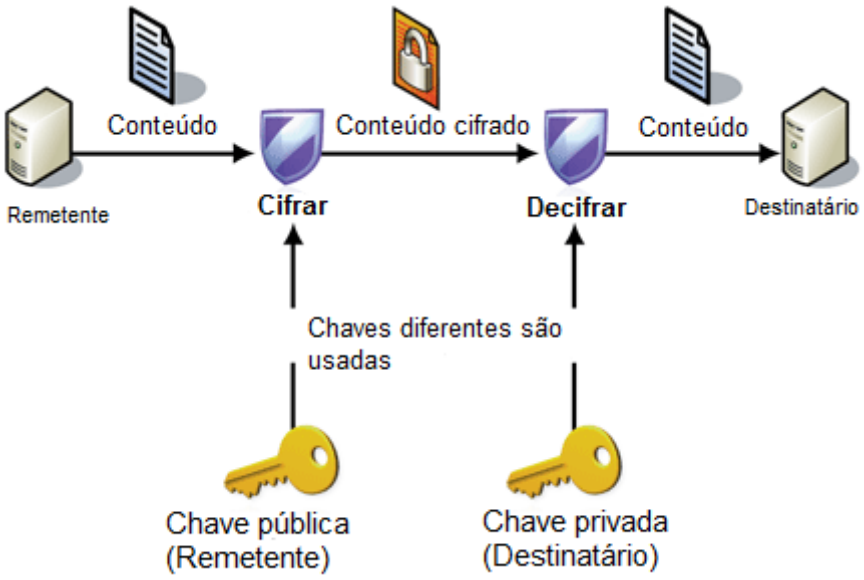

Figura 4.2: Figura com o diagrama de uso de criptografia de chaves assimétricas [26](adaptada para o português).

4.2 Certificados

Uma característica do TLS/SSL é garantir autênticidade das partes envolvidas. Para isso, o protocolo exige um certificado por parte do servidor usando TLS e, opcionalmente, do cliente. Um certificado é um arquivo contendo uma chave pública, dados da organização dona do certificado, dados da autoridade certificadora e informações sobre o tipo de criptografia usada.

O certificado é enviado a quem deseja se conectar seguramente ao servidor, o cliente irá verificar a veracidade desse certificado com uma autoridade certificadora. Depois o cliente trocará algum tipo de mensagem cifrada com o servidor, pois apenas quem possuir a chave privada poderá entender aquela mensagem, logo se o servidor responder corretamente ele é dono da chave privada corrreta. Se o servidor é de fato dono da chave privada, pode-se garantir a sua identidade baseado na autoridade certificadora.

Autoridades certificadoras costumam cobrar pelos seu serviços, por isso acabaram por se popularizar certificados auto assinados. Esses certificados não possuem nenhuma autoridade certificadora, um cliente deve confiar cegamente nesse certificado para proceder com a conexão. Na figura 4.3 podemos ver um certificado auto assinado, no topo da figura há um aviso aos usuários de que o certificado não possue uma autoridade certificadora e que o usuário está confiando de livre e espontânea vontade naquele certificado.

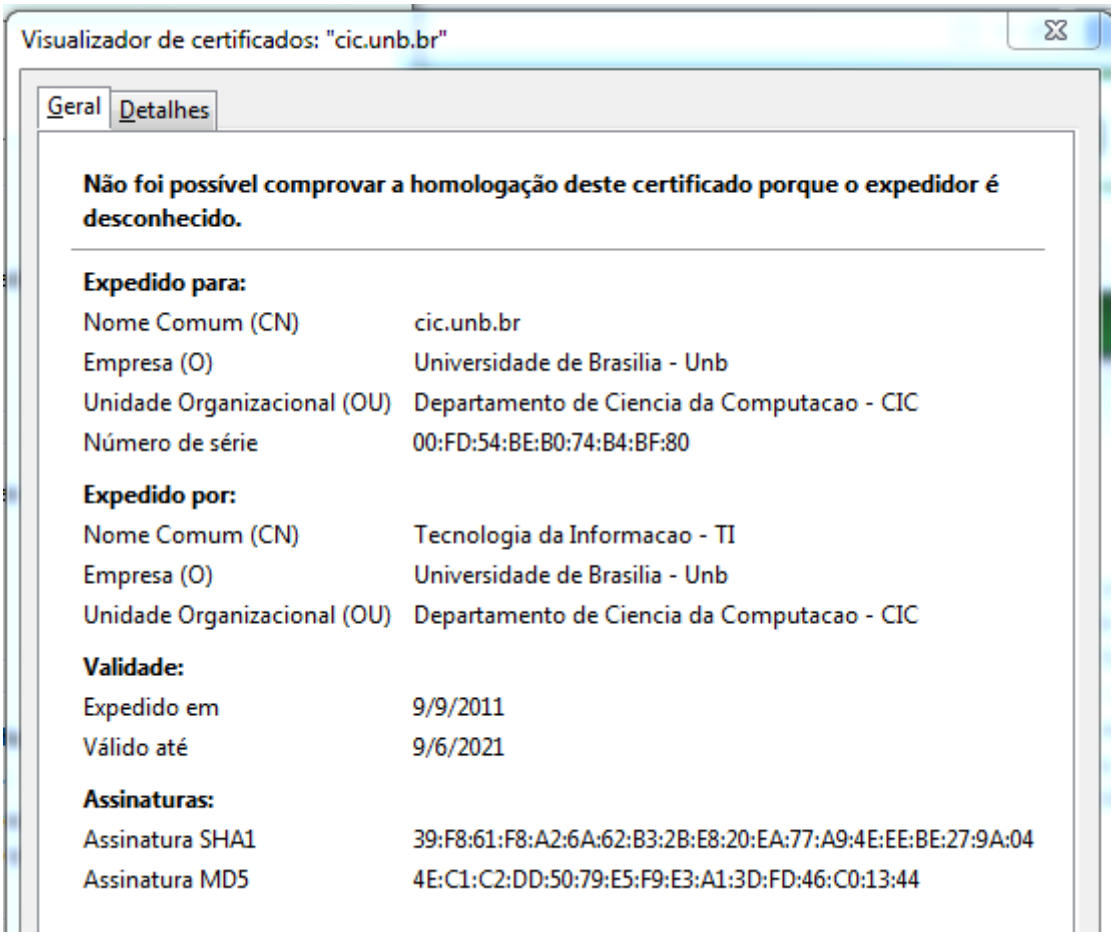

Figura 4.3: Exemplo de um certificado auto assinado e de campos de informação dos certificados.

4.3 Funções de resumo

Outra característica do protocolo TLS é garantir a integridade das mensagens, garantir que as mensagens enviadas sejam iguais as mensagens recebidas. Com essa finalidade usam-se funções de resumo (no inglês *hash*), essas funções recebem como entrada uma grande quantidade de bytes e retornam uma quantidade menor e diferente para cada entrada. As funções de *hash* retornam uma quantidade de bytes que equivale a uma 'identidade' da entrada, mas de menor tamanho. Boas funções de *hash* tem uma saida diferente para cada entrada e é impossível obter a mensagem original a partir de sua saída.

Existem várias funcões de resumo criptográfico, no protocolo TLS são usadas para garantir que a mensagem esteja íntegra. Cada mensagem é concatenada com seu *hash*, assim, quem recebe a mensagem tira o seu próprio *hash* e compara com o valor concatenado na mensagem. Caso os valores coincidam, a mensagem está correta, caso haja uma divergência, houve modificações ou irregularidades na mensagem.

4.4 *Handshake* TLS/SSL

Uma vez que o *handshake* seja concluído toda troca de informação será criptografada e o *handshake* é que define como isso será feito [9] [8]. O processo de *handshake* começa sempre pelo cliente, esse enviará uma mensagem no formato TLS informando que quer se conectar seguramente com o servidor. A primeira mensagem é denominada *client hello*, ela contêm até qual versão o cliente suporta o TLS/SSL, uma identidade de sessão (caso esteja retomando uma outra conexão), bytes aleatórios, um conjunto de combinações de cifras e *hashs* disponíveis e um conjunto de métodos de compressão. Mais detalhes podem ser observados na figura 4.4 abaixo.

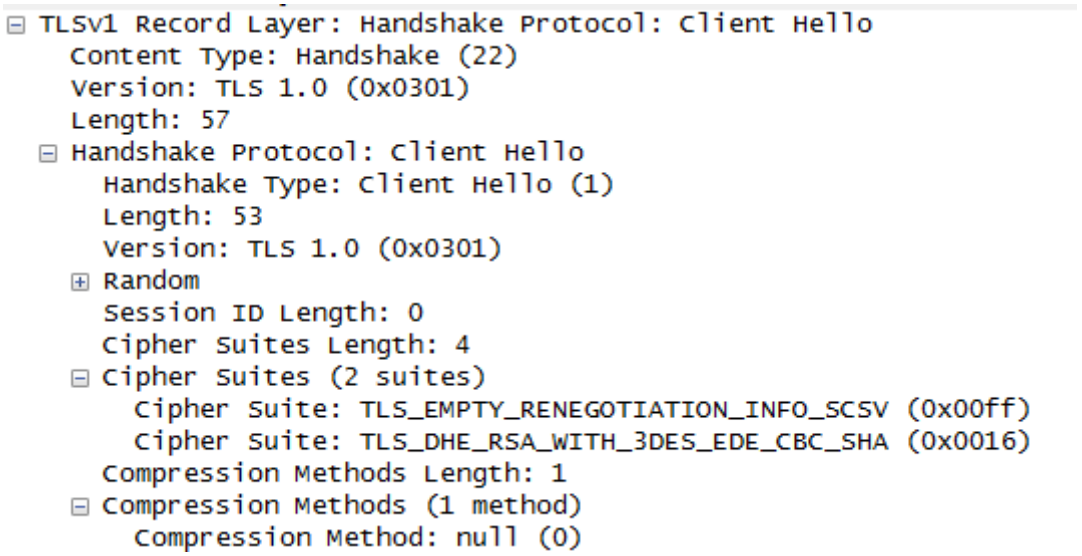

Figura 4.4: Exemplo de *client hello*, pode-se observar que o campo de sessão está vazio pois é uma nova sessão.

No *client hello* são enviados um conjunto de opções de cifras, cada uma dessas opções de cifras é denominada uma *cipher suite*. Cada *cipher suite* diz como será feita a troca de chaves entre o cliente e o servidor, qual será o método de cifragem e qual será a função de resumo usada [32].

Tabela 4.1: Exemplos de *cipher suites* [17].

Cipher Suite	Comentário
TLS_RSA_WITH_RC4_128_MD5	Essa *suite* usa RSA para negociar as chaves, usa RC4 como método de cifra e a função de *hash* MD5.
TLS_RSA_WITH_RC4_128_SHA	Essa *suite* é similar a primeira mas usa como *hash* a função SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	Essa *suite* usa o protocolo Diffie & Hellman efêmero para troca de chaves diferente das outras.
TLS_RSA_WITH_DES_CBC_SHA	
TLS_DHE_DSS_WITH_DES_CBC_SHA	
TLS_RSA_EXPORT1024_WITH_RC4_56_SHA	
TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA	
TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA	
TLS_RSA_EXPORT_WITH_RC4_40_MD5	
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	
TLS_RSA_WITH_NULL_MD5	
TLS_RSA_WITH_NULL_SHA	

As opções de compressão no final do *client hello*, figura 4.4 são raramente usadas.

Depois do cliente mandar o *client hello* com as *cipher suites*, o servidor responde com um *server hello*, esse pacote contém qual versão do TLS o servidor irá usar, junto com a *cipher suite* escolhida, uma identidade de sessão, o método de compressão escolhido e bytes aleatórios.

```
⊟ SSLv3 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: SSL 3.0 (0x0300)
    Length: 74
  ⊟ Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 70
      Version: SSL 3.0 (0x0300)
    ⊞ Random
      Session ID Length: 32
      Session ID: a085a9b1395f17fc8a66a834bb4de5ca038aa7fc1de159b3...
      Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
      Compression Method: null (0)
```

Figura 4.5: Exemplo de *server hello*.

Junto com o *server hello* o servidor também manda uma mensagem com o seu certificado e uma mensagem opcional (*server key exchange*) usada somente em protocolos de

troca de chave Diffie & Hellman (DH e DHE). Para finalizar, o servidor também manda um *server hello done* que apenas serve para simbolizar o fim da sua ação. No geral, a mensagem enviada pelo servidor possue o *server hello*, o certificado e o *server hello done* todos juntos, como na figura 4.6.

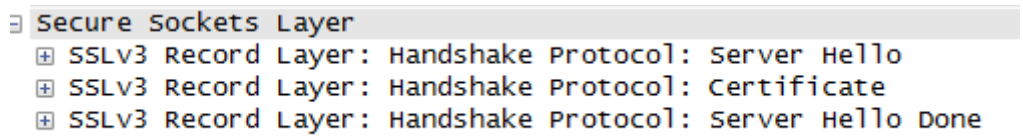

Figura 4.6: Exemplo completo, com todas as mensagens necessárias na resposta do servidor.

Finalizando o *handshake* o cliente responde com três mensagens, um *client key exchange*, um *change cipher spec* e um *encrypted handshake message*.

O *client key exchange* serve para terminar a negociação das chaves, no RSA é enviado um segredo com a chave que somente o servidor poderá decifrar, no Diffie & Hellman é enviado uma chave pública que será usada pelo servidor para computar a chave da sessão, detalhes mais específicos serão mostrados em subseções futuras.

O *change cipher spec* é apenas um byte dizendo que todo o fluxo de mensagem a partir dele será cifrado. E por último, o *encrypted handshake message* é o *hash* de todo o processo de *handshake* já cifrado, essa mensagem será decifrada pelo servidor e o seu *hash* será conferido, em caso positivo o servidor responde, também, com um *change cipher spec* e um *encrypted handshake message*. Feito todo esse processo, o *handshake* está completo e todo tráfego entre cliente e servidor será cifrado garantindo confidencialidade e integridade(*hash*).

Figura 4.7: Exemplo com a resposta final do cliente.

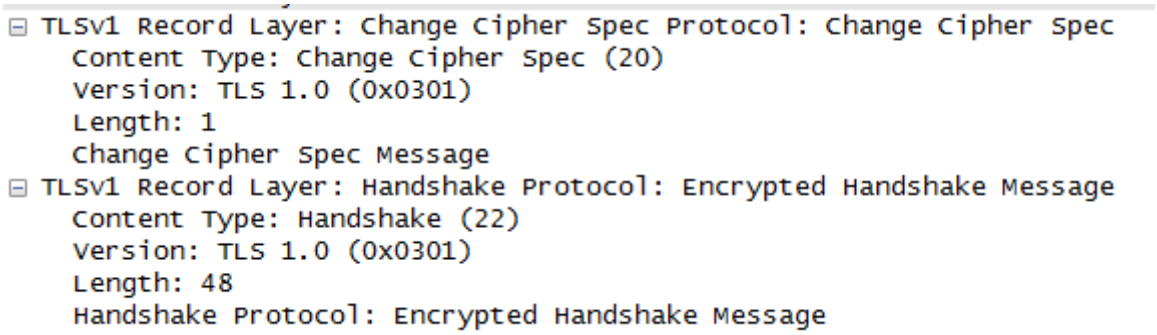

Figura 4.8: Exemplo com a resposta final do servidor, finalizando o *handshake*.

Em resumo, o cliente inicia o *handshake* com um *client hello*, o servidor responde com pelo menos três mensagens (*server hello*, seu certificado, *server hello done*). Nessa resposta, o servidor também pode pedir um certificado do cliente, mas esse fato é raro e não é necessário para esse documento. O cliente então encerra o processo e dá abertura a conexão cifrada, ele envia um *client key exchange*, um *change cipher spec* e um *encrypted handshake message*. O servidor confere essas últimas mensagens e responde igualmente com um *change cipher spec* e um *encrypted handshake message*. Ambas as partes estabelecem assim o fim do *handshake* e o ínício da conexão segura. Caso qualquer etapa do *handshake* falhe, um alerta é enviado pelo servidor.

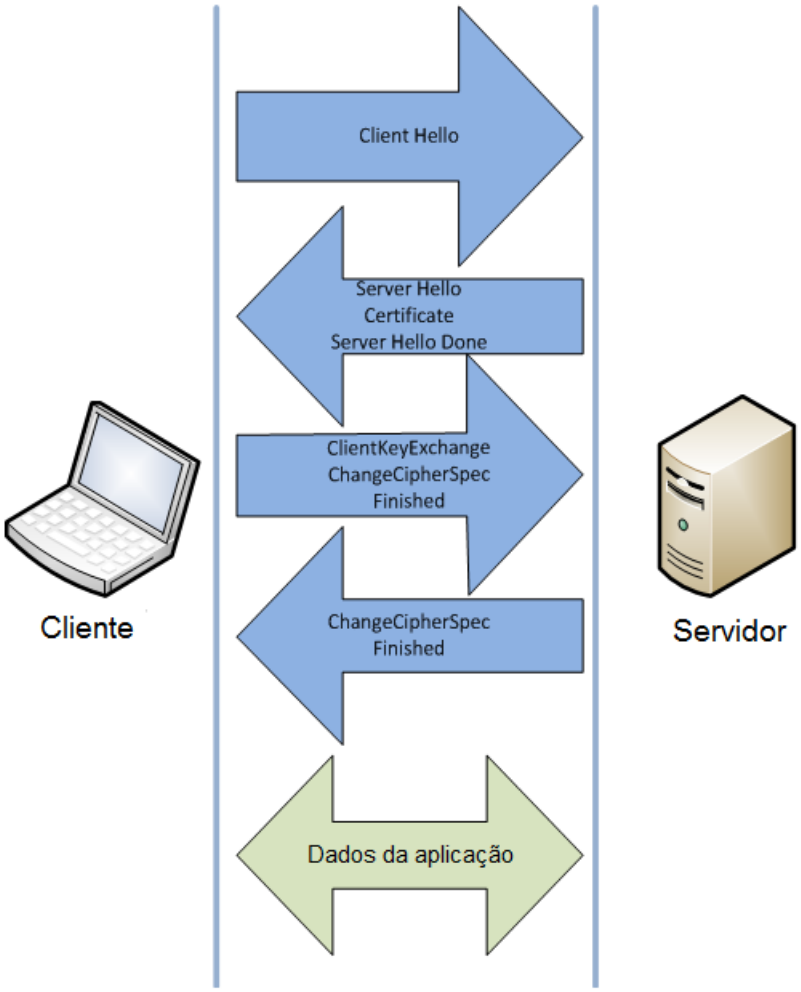

Figura 4.9: Exemplo geral de um *handshake*. Nota-se que o *encrypted handshake message* está sendo chamando apenas de *finished*.

4.5 Formato dos pacotes TLS/SSL

Para a confecção da ferramenta deste trabalho de conclusão de curso foi preciso um conhecimento detalhado dos pacotes TLS/SSL, pois a ferramenta desenvolvida usa abundantemente desses pacotes e cada um dos bytes de cada pacote da ferramenta são manualmente atribuídos de acordo com as regras de formação de pacotes TLS/SSL [9] [8].

Cada pacote TLS/SSL começa com um byte dizendo se o pacote faz parte do *handshake*, é um alerta, um *change cipher spec* (rever seção anterior) ou um pacote normal com o TLS já em andamento. Logo em seguida há dois bytes com a versão do protocolo usado e mais dois bytes com o tamanho da mensagem.

Tabela 4.2: Tipos de pacotes TLS/SSL [16].

Byte(hexadecimal)	Nome	Descrição
0x14	ChangeCipherSpec	Indica que as próximas mensagens estarão cifradas.
0x15	Alert	Um alerta de algo errado durante a execução do protocolo.
0x16	Handshake	Característico de todas as mensagens do *handshake*.
0x17	Application	Presente nas mensagens depois que o protocolo já está ativo.

Logo em seguida, há um campo detalhando a funcionalidade da mensagem em si, caso ela seja um alerta ou parte do *handshake*. No caso de alertas, há dois bytes especificando se o alerta foi fatal ao uso do protocolo ou só uma precaução e o código do erro que ocorreu. Caso o pacote seja parte do *handshake*, há um byte que diz qual o tipo da mensagem em si, como pode ser visto na tabela 4.3 abaixo.

Tabela 4.3: Tipos de mensagens durante o *handshake* [16].

Byte(decimal)	Nome
0	HelloRequest
1	ClientHello
2	ServerHello
11	Certificate
12	ServerKeyExchange
13	CertificateRequest
14	ServerHelloDone
15	CertificateVerify
16	ClientKeyExchange
20	EncriptedHandshakeMessage

O formato das mensagens com o protocolo em funcionamento são bem simples, como já dito, há um byte para o tipo (*application*) , a versão do protocolo, o tamanho do resto do pacote e os dados cifrados. Já os pacotes referentes ao *handshake* são bem variados mas tendem a manter a estrutura da figura 4.10.

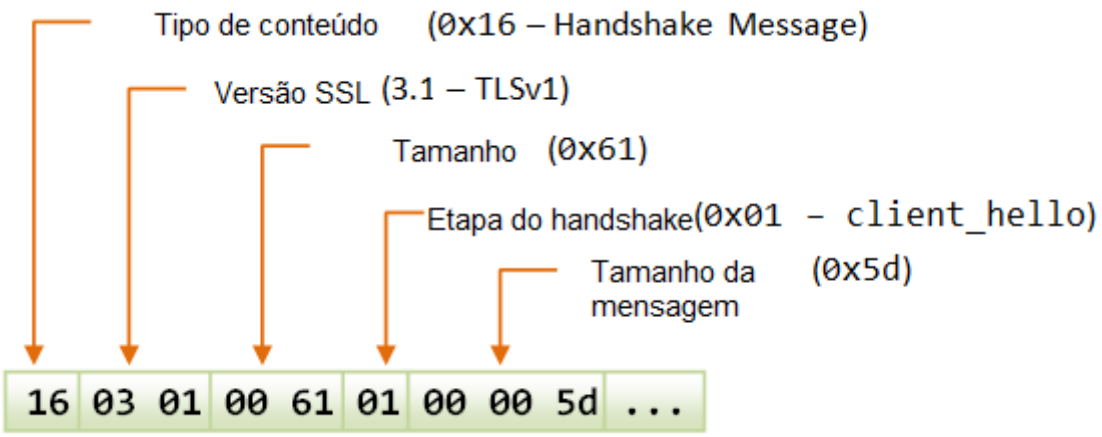

Figura 4.10: Esquema geral de um pacote do protocolo TLS/SSL durante um *handshake* [16](adaptada para o português).

4.6 Diffie & Hellman

O Diffie & Hellman é um método de negociação de chaves, as duas partes envolvidas transmitem parâmetros entre si e deduzem a partir deles uma mesma chave. Um possível atacante observando todos os parâmetros transmitidos não consegue deduzir a mesma chave, o método é computacionalmente seguro. Esse método está implementado dentro de várias *cipher suites* e com várias adaptações disponíveis.

Na confecção da ferramenta associada a esse documento, é usado o protocolo em sua versão básica. Por tal motivo, se torna necessário um entendimento razoável do seu funcionamento. Usando como exemplo um cliente e um servidor, podemos dizer que o protocolo começa com ambos escolhendo um número secreto. O servidor escolhe o número 'a' e o cliente o número 'b', em seguida o servidor escolhe uma raiz primitiva[1] 'g' e um número primo 'p'.

De posse desses valores, o servidor envia ao cliente 'A', o resultado da seguinte equação:

$$A = g^a \bmod p \tag{4.1}$$

O cliente pega o resultado da equação 4.1 e eleva a seu número 'b' módulo 'p', assim obtento a chave 'K'.

$$K = A^b \bmod p \tag{4.2}$$

O cliente então manda para o servidor o resultado 'B' da seguinte equação:

$$B = g^b \bmod p \tag{4.3}$$

Assim, o servidor chega no mesmo valor de chave 'K' através da equação:

$$K = B^a \bmod p \tag{4.4}$$

[1]Basicamente, uma raíz primitiva módulo n é um número que quando elevado a um expoente inteiro gera todos os números daquele espaço n, ou seja, todos os números de 1 até n podem ser optidos exponenciando a raíz a algum expoente inteiro modulo n.

No final do processo, ambos servidor e cliente possuem a mesma chave e os únicos valores que precisaram ser transmitidos foram 'A','g','p' e 'B'. Com apenas esses valores fica computacionalmente inviável a um observador descobrir a chave 'K'. A figura 4.11 ilustra de forma bem simples o método.

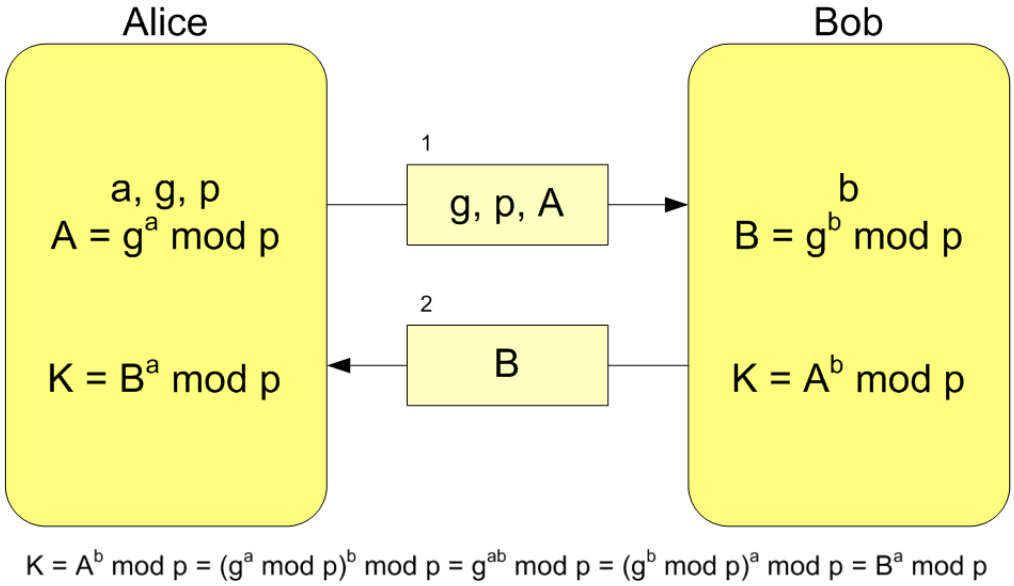

Figura 4.11: Diagrama da negociação de chaves usando Diffie & Hellman [31]. Sendo 'K' a chave, 'g' o gerador, 'p' um número primo e 'a' e 'b' números escolhidos aleatóriamente.

Capítulo 5

Motivações

Este capítulo contém as motivações e inspirações desse documento, seu objetivo é mostrar ataques relacionados e a situação atual da segurança de redes em relação a ataques de negação de serviço envolvendo SSL/TLS. Serão mostrados ataques que inspiraram a confecção do novo ataque e expectativas de possíveis resultados e impactos baseado na situação atual da segurança digital.

5.1 Objetivos e Código de Conduta

Como já mencionado, a confecção das ferramentas de ataque deste trabalho não tem como objetivo causar mau, e sim verificar a possibilidade do ataque para assim alertar aos interessados sobre sua existência. Muitas vezes é difícil prever novas formas de ataque de negação de serviço, no geral, é necessário primeiro conhecer o funcionamento do ataque para assim planejar uma solução ou mecanismo de defesa. Esse documento tem o objetivo de facilitar a construção desse mecanismo de defesa, mostrando abertamente como um ataque de negação serviço usando TLS/SSL funcionaria em diferentes estados criptográficos, inclusive o *Perfect Foward Secrecy*.

Nos próximos capítulos, pretende-se mostrar claramente detalhes da implementação do ataque e como seus testes foram realizados, para assim deixar claro seu funcionamento, seus efeitos e impactos nos servidores. Para tentar impedir o uso incorreto do ataque, todos os artefatos e códigos gerados para a confecção desse documento serão mantidos confidencialmente entre o autor e o seu orientador.

Em suma, o objetivo deste documento é o desenvolvimento de ferramentas que causem um ataque de negação de serviço contra servidores *web*, analisando todas as possibilidades de ataque por exaustão de recursos computacionais. Para exaurir esses recursos, serão usadas conexões seguras(TLS/SSL), mostrando que mesmo conexões seguras podem ser a causa de um ataque, não sendo totalmente confiáveis. Os ataques, então, serão testados em ambiente controlado(fugindo do enquadramento de crime cibernético) contra servidores com TLS/SSL. Os resultados servirão para avaliar a viabilidade do ataque demonstrando sua funcionalidade e medindo seus impactos no cliente e no servidor.

5.2 Ataques relacionados

Uma das inspirações deste projeto foram ataques passados que usavam de SSL. Um desses ataques e um dos primeiros a usar SSL foi o ataque de renegociação de chaves. Esse ataque abusava de uma opção do protocolo TLS/SSL e do custo computacional associado a ela [5].

Como mostrado em capítulos anteriores, cliente e servidor podem gerar chaves de sessão usando diferentes mecanismos, porém essas chaves são geradas durante a criação da conexão. A geração dessas chaves gera um custo computacional devido ao trabalho criptográfico envolvido, e normalmente só ocorre uma vez, durante a criação da conexão segura. Mas como previsto nos protocolos, o cliente e o servidor envolvidos numa conexão segura usando TLS/SSL podem renegociar chaves. O ataque consistia de uma sequência de renegociações de chaves [5], onde o cliente não calculava sua chave mas exigia do servidor sua nova chave. Essas sucessivas renegociações exigiam muito do servidor e causavam exaustão, numa renegociação normal o gasto computacional do servidor é 10 vezes maior que no cliente.

Pode-se perceber que o ataque usa de uma opção viável pelo protocolo para causar a negação de serviço, tal como pretende-se fazer com as ferramentas criadas nesse documento [7]. Esse ataque é praticamente obsoleto, pois uma solução foi amplamente adotada pelos servidores da internet. Embora o protocolo TLS/SSL não tenha sido alterado, os fabricantes de software de servidores *web* criaram atualizações nas quais clientes não mais poderiam renegociar chaves, apenas servidores. Essas atualizações diminuiram a força do ataque, provando-se uma solução eficiente, tal qual pretende-se inspirar com este documento.

Um outro ataque bem similar ao descrito, foi o ataque proposto por Vincent Bernat em seu blog [33]. Estudando técnicas para mitigação do ataque de renegociação de chaves, o autor percebeu que o custo computacional de decifrar uma mensagem pelo servidor é mais caro que o custo de cifrar. É de conhecimento popular em criptografia assimétrica que o custo de cifrar e decifrar variam, mas devido ao funcionamento dos protocolos de segurança e a forma como certificados funcionam, o servidor quase sempre terá um maior custo para decifrar. O autor então sugeriu a possibilidade de criar um ataque onde o cliente não cifra suas mensagens e sim manda um 'lixo' qualquer. O servidor tem que decifrar a mensagem para então perceber que se trata de 'lixo', logo o cliente não tem custo e o servidor arca com o custo de decifrar.

Espera-se que usando algumas das sugestões propostas por Vincent Bernat [33] que se possa confeccionar ferramentas para o ataque. A idéia do ataque proposto neste documento consiste em mandar mensagens com 'lixo' ainda durante o *handshake* TLS/SSL, para que o servidor tenha uma maior custo. Partindo da idéia geral dos DoS, a vítima tem que consumir mais recursos do que o cliente gasta para atacá-lo. Propõe-se, então, a construção de uma ferramenta que usa desse desequilíbrio, para com um baixo custo no cliente causar um grande consumo no servidor e consequentemente o DoS.

Além desses conceitos pretende-se também avaliar o uso de diferentes *cipher suites*, testando quais possuem um maior custo para o servidor, assim aumentando o desequilíbrio. Comparando as diversas *cipher suites*, verifica-se que um dos mais populares protocolos de negociação de chaves é o RSA, porém o protocolo estava envolvido em recentes polêmicas envolvendo a NSA(Agência de Segurança Norte Americana) [13] [20].

O último escândalo envolvendo a NSA deixou transparecer que a agência estava espionando cidadões e empresas na internet, monitorando e gravando suas conexões, descobrindo informações privadas. Estudiosos de criptografia então começaram a recomendar o uso de conexões seguras, porém ficou claro que se a agência americana estivesse de fato armazenando dados transmitidos, o protocolo RSA não seria mais confiável [20] [29]. O protocolo RSA não seria mais seguro pois no protocolo são usados um par de chaves que não costuma variar, por mais que as chaves sejam computacionalmente seguras, a NSA usando vários computadores pode dentro de um longo período quebrar a chave. Mesmo que a quebrar das chaves demore, uma vez quebrada a NSA poderia decifrar todas as mensagens que já possue armazenadas.

Com essas preocupações em mente, o *Perfect Foward Secrecy*(segredo perfeito) ganhou um lugar especial na comunidade. O *foward secrecy* consiste em usar uma chave diferente para cada conexão, para isso usam-se *cipher suites* que usem do protocolo Diffie & Hellman efêmero, tanto na versão normal quanto na versão com curvas elípticas, pois ele garante que as chaves serão transmitidas seguramente entre cliente e servidor. Com uma chave diferente para cada conexão a NSA ou outro possível atacante teria que quebrar as chaves de cada conexão, o que seria computacionalmente inviável [20] [29].

Mas o *Perfect Foward Secrecy* também envolve um maior custo computacional já que chaves devem ser geradas e negociadas a cada conexão, logo esse custo computacional extra também pode ser usado por um atacante, como neste documento. Nos testes das ferramentas de ataque pretende-se testar o uso do *Perfect Foward Secrecy* para mostrar que seu uso não é a solução perfeita pois o mesmo traz desvantagens [29] [20].

Tendo-se em mente todos essas ferramentas prévias e o conhecimento desses fatos, planejou-se esse documento com a intenção de testar a possibilidade de ataques de negação de serviço envolvendo TLS/SSL como contribuição para a área de segurança de redes. Criando uma vantagem para a área de segurança antes que ataques desse tipo começem a acontecer.

5.3 Contribuições e contrastes

A contribuição deste documento, em suma, é testar novas estratégias de ataque que podem desencadear um novo ataque de negação de serviço, analisá-lo, expor os resultados e mostrar conclusões a respeito de sua funcionalidade, possíveis idéias de mecanismos de defesa e outras melhorias que um possível atacante pode adotar.

A proposta do novo ataque difere de ataques semelhantes pois consiste da combinação desses ataques junto com melhorias, em busca de um melhor desempenho. Combinando as idéias de Vincet Bernat [33], ataques de DoS famosos, *benchmarks* de *cipher suites* e o ataque de renegociação de chaves TLS/SSL, pretende-se criar um ataque que seja viável e funcional em servidores *web* atuais.

A primeira idéia para o ataque é usar conexões completas, no ataque de renegociação de chaves a conexão é reiniciada antes de ser completada, pois o cliente requere uma nova chave propositalmente. Como a renegociação não é mais possível por parte do cliente, a solução planejada é deixar a conexão ser estabelecida e então refazê-la do início, ao invés de requerer uma renegociação.

Outra diferença no ataque planejado é tentar estabelecer uma maior diferença entre o consumo de CPU do cliente e do servidor. Caso o consumo de CPU do cliente seja bem

menor que o do servidor, menos clientes (e possivelmente menor poder de processamento) serão necessários para exaurir o servidor. Esse desnível presente em muitos ataques de DoS é essencial pois servidores geralmente são robustos quando comparados a computadores pessoais (atacantes).

Para alcançar esse desnível pretende-se usar a idéia de Vincent Bernat em seu blog [33], mandar a mensagem final (*encrypted handshake message*) do processo de *handshake* com 'lixo'. A mensagem final do *handshake* consiste de um resumo cifrado de todo o *handshake*, o servidor precisa decifrar a mensagem (gasto computacional) para ver se o resumo confere, mas o gasto computacional de criar *bytes* aleatórios(lixo) pelo cliente é zero. Um maior desnível entre os gastos de CPU deve ser alcançado usando essa idéia e testando cifras que sejam mais onerosas computacionalmente. Outras estratégias também serão avaliadas em busca de tornar o ataque mais eficiente para que possa ser testado.

Além das contribuições citadas, pretende-se realizar testes de versões do ataque com *Perfect Foward Secrecy*. Como o uso do protocolo Diffie & Hellman efêmero começou a ser sugerido e enaltecido pela sua maior segurança [20], o mesmo será testado para avaliar sua eficácia e segurança em relação aos ataques de DoS.

Capítulo 6

Implementação e Testes

O presente capítulo trata de detalhes da implementação das ferramentas de ataque e como foram realizados os testes. São abordados os aspectos e escolhas que levaram as ferramentas à sua forma final, detalhes de como foram testados os ataques e quais ferramentas foram usadas para os testes.

6.1 Implementação

Como prova de conceito do ataque, foram confeccionadas várias ferramentas e para a confecção dessas ferramentas de ataque foram feitos vários protótipos e aprimoramentos até que as ferramentas escolhidas chegassem em seu estado final. Muitos testes foram realizados deixando claro que alguns métodos não seriam eficientes como ferramenta e por consequência chegando-se a um concenso de quais seriam as mais adequadas para um teste completo. As ferramentas que serão citadas nesta seção são apenas referentes aos produtos finais, para não alongar desnecessariamente o seu conteúdo.

Foi escolhido como linguagem de programação, C/C++. A escolha vem da rapidez no tempo de execução e da portabilidade do código para vários sistemas operacionais. Além disso umas das ferramentas usa o aplicação multiplataforma e U.I. framework QT® [28], escolhida pelo enorme acervo de bibliotecas(incluindo bibliotecas para o uso de funções de rede), facilidade para o design e criação de interfaces gráficas. Usando QT® [28] pode-se programar em C/C++ com um pequeno overhead para enormes vantagens e ainda mantendo a portabilidade. A aplicação/framework inclusive possui funções para sockets já implementadas e threads, que foram usadas para agilizar o processo de simular vários clientes em uma única máquina atacante.

Para garantir que o uso da ferramenta com QT®não estivesse com um overhead significativo, foi feito uma outra ferramenta similar mas usando POSIX threads [2], uma API seguindo o padrão POSIX que permite o uso de threads, muito popular entre os usuários de sistemas UNIX. Em vários testes, ambas as ferramentas com POSIX threads e QT®tiveram o mesmo desempenho, optou-se então em manter a ferramenta usando QT®e abandonar as POSIX threads.

Outras ferramentas acabaram por usar a API libevent [24] que mostrou-se bem eficiente. A API trata eventos relacionados a funções do sistema operacional sendo bem

eficiente em lidar com múltiplas instâncias[1] de eventos envolvendo chamadas ao sistema. Como as ferramentas em si precisam de múltiplas chamadas ao sistema operacional, pois usam funções de rede quase que exclusivamente, a API por mais complicada que fosse mostrou-se um par perfeito para as ferramentas. A sugestão de usar a API veio de comentários e sugestões recebidas no *blog* de Vincent Bernat [33], já citado anteriormente.

Para os *handshakes* TLS/SSL não foi usada nenhuma biblioteca criptográfica, todos os pacotes foram confeccionados de forma semi-automatica e *byte* a *byte* adequando-se as necessidades das vítimas.

Antes de entrar no mérito do resultado final das ferramentas, é desejado relembrar que esse documento tem propósitos acadêmicos apenas e não pretende causar mal, e sim ajudar. Tendo isso em mente, reforça-se que nenhum artefato será disponibilizado para a comunidade, eles serão mantidos confidencialmente entre autor e orientador.

Quatro ferramentas obtiveram maior destaque e foram consideradas mais relevantes para os testes envolvendo servidores reais. Duas das ferramentas foram criadas usando a API libevent [24], a API trabalha baseada em eventos, então, sempre que um pacote é recebido do servidor(evento) a ferramenta responde enviando o próximo pacote (próximo evento). Esse comportamento é idêntico ao comportamento real onde as mensagens entre cliente e servidor são intercalados entre si até o final do handshake. A diferença entre as duas ferramentas usando da API libevent está no fato de uma usar apenas *ciphersuites* baseadas em RSA e a outra em Diffie & Hellman, escolhidas justamente para mostrar a diferença de força entre os ataques e averiguar se o *perfect foward secrecy* é realmente mais seguro ou na verdade mais perigoso (rever capítulo anterior).

As outras duas ferramentas foram implementadas usando o framework QT®e suas threads [28]. A primeira ferramenta faz com que suas threads esperem uma resposta do servidor para então respondê-la, assim como o protocolo descreve. A segunda versão da ferramenta faz com que as mensagens dos *handshakes* sejam mandadas o mais rápido possível, possuindo um caráter mais volumétrico. Essa segunda versão verifica se a conexão TCP(acima da conexão SSL) está ativa, em caso afirmativo ele manda a continuação das mensagens sem esperar que o servidor mande a resposta. Esse comportamento garante que o servidor ficará sempre ocupado pois as mensagens do cliente chegam uma após a outra. Caso a conexão tenha sido encerrada por qualquer motivo a conexão é restabelecida e o processo continua. Essa segunda abordagem ganhou destaque pois mesmo que o servidor não fique com uma conexão ociosa esperando o cliente, ele recebe mais *handshakes* 'incorretos' e logo gasta maior poder computacional decifrando.

[1] A API usa de processos de escalonamento para simular várias execuções, mas não são execuções reais, sendo diferentes de *threads* ou novos processos. A multiplicidade simulada pela API será denominada por termos como clientes/instâncias.

Tabela 6.1: Descrição abreviada das diferenças entre as quatro ferramentas.

Tabela de Ferramentas			
Libevent		QT®	
RSA	DHE	Versão 1	Versão 2
Usa apenas *ciphersuites* que usam RSA como protocolo de acordo de chaves	Usa *ciphersuites* que usam de Diffie&Hellman	Essa versão sempre espera uma mensagem do servidor para responder, seguindo a lógica normal do protocolo	Essa versão manda respostas e requisições o mais rápido possível e sempre que possível

Em resumo, 4 das ferramentas desenvolvidas se destacaram em testes simplórios e avançaram para os testes envolvendo servidores reais. Para evitar confusões futuras as 4 ferramentas destacadas serão mencionadas da seguinte forma: primeira versão (usando QT®e threads), segunda versão(usando QT®, threads e o comportamento impaciente), versão com libevent e RSA, versão com libevent e DHE(Diffie & Hellman efêmero). Para maior facilidade, as peculiaridades das ferramentas estão resumidas na tabela 6.1 acima.

Todas as quatro ferramentas descritas, fora suas peculiaridades, têm o mesmo comportamento geral. A primeira parte desse comportamento geral das ferramentas é testar se o endereço fornecido a elas corresponde a um servidor e se esse servidor aceita conexões seguras TLS/SSL. Isso é feito mandando a primeira mensagem do *handshake* TLS/SSL que está gravada no código do programa, caso o servidor responda e sem um alerta ele é um alvo viável para o ataque.

A segunda parte da rotina é verificar quais *cipher suites* são aceitas pelo servidor, de preferência escolher a mais lenta. Dentre uma lista de prioridades de *cipher suites* já testadas, a ferramenta escolhe a mais lenta suportada pelo servidor. Essa etapa é feita mandando vários primeiros pacotes de *handshake* com apenas uma *cipher suite* cada, então o servidor responde com um alerta às quais não suporta. Essas *cipher suites* foram testadas previamente[2] e já estão no código da ferramenta, pois uma leitura delas junto com o ataque poderia não ser fiél devido a instabilidades momentâneas e além disso deixaria o ataque mais evidente. Mais detalhes na figura 6.1 abaixo.

[2] As *ciphersuites* mais onerosas foram escolhidas de acordo com o *benchmark* proposto por Vincent Bernat em seu blog [33].

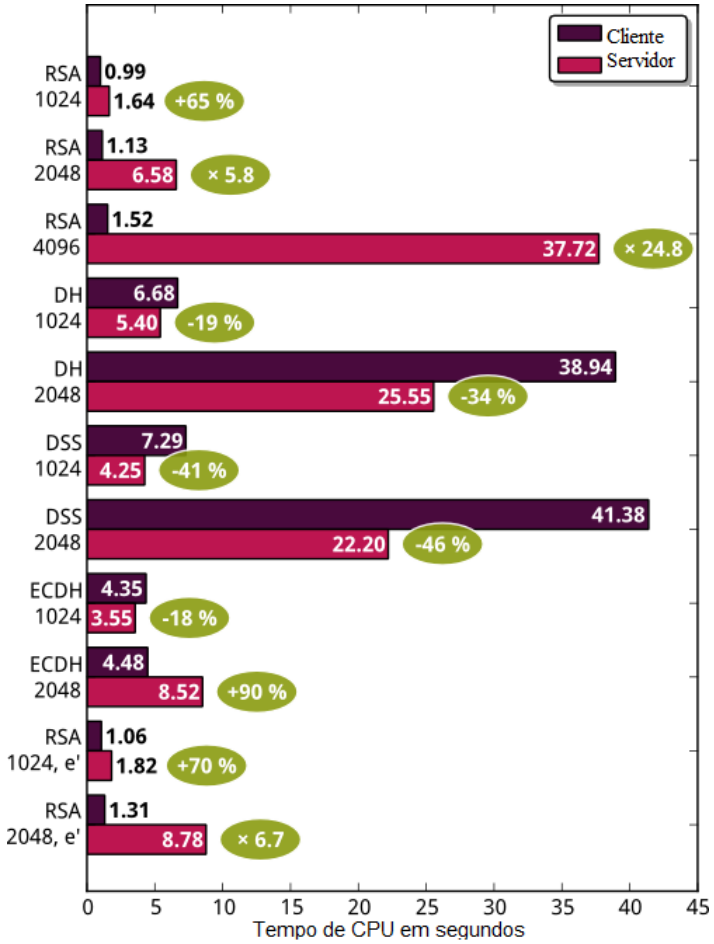

Figura 6.1: Imagem contendo o *benchmark* de *ciphersuites* usado como parâmetro [33] (adaptada para o português).

Em seguida as ferramentas montam os pacotes que serão usados para resposta, os pacotes já estão semi-prontos no código só precisam de detalhes simples como tamanho de chaves e etc. Os pacotes então são levemente adaptados, as *timestamps* dos pacotes são colocadas com a mesma data para agilizar o processo, o pacote com 'lixo' é montado usando valores aleatórios e do tamanho correto(os prévios *handshakes* disponibilizam esse dado).

Para finalizar e começar a negação de serviço em si, são disparadas as threads ou clientes. Cada thread ou cliente possue o mesmo comportamento, dispara *handshakes* atrás de *handshakes*, usando os pacotes já montados na etapa anterior. O processo continua até que o usuário resolva interroper o ataque.

O comportamento geral das ferramentas está resumido na figura 6.2 abaixo, todas as 4 ferramentas seguem o fluxograma mostrado mas mantendo a diferenças citadas.

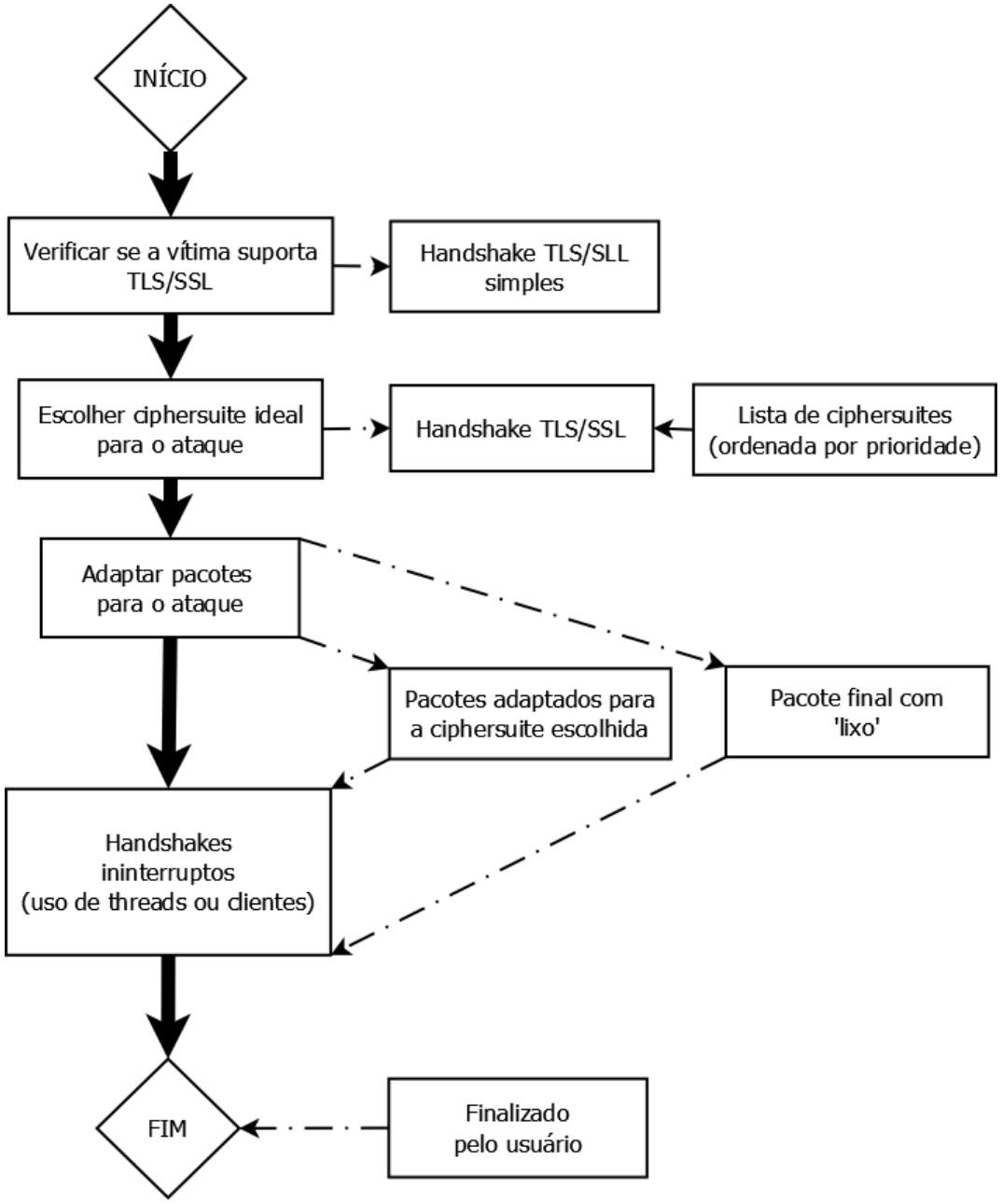

Figura 6.2: Imagem contendo o fluxograma de como as ferramentas funcionam. Diferenças bem sutis diferenciam as ferramentas, mas o funcinamento padrão é o esquematizado.

6.2 Ferramentas e metodologia usadas nos testes

Os principais objetos de teste são o atacante, o cliente e o servidor. O cliente é o usuário legítimo, que sofrerá a degradação de serviço ou sua completa negação, por isso é importante mensurar o impacto sofrido, além disso monitoramento no servidor é necessário para medir a intensidade do ataque e como o ataque se desenvolve para causar a negação de serviço.

Para medir o efeito do ataque no servidor, o mesmo terá seu gasto de CPU monitorado, junto com a quantidade de conexões ocupadas e ociosas. Como o cliente interage com o servidor apenas para fazer requisições e receber respostas, o cliente será monitorado para o aumento do tempo de resposta de suas requisições. Por exemplo, um cliente requisita uma página *web* e essa página demora cerca de alguns milisegundos para ser entregue. Assim, durante o ataque será monitorado o acrécimo desse tempo de resposta.

Os testes forão todos realizados em ambiente isolado para se ter maior controle, menor interferência e evitar problemas legais, pois ataques de negação de serviço podem ser enquadrados como crime cibernético. O ambiente de teste consiste de três máquinas, um servidor, um atacante e o cliente. Essas três máquinas serão ligadas diretamente usando um roteador a/b/g/n e cabos, para assim obter-se menor atraso nos pacotes, eliminar tráfego externo e executar uma melhor medição. Um ataque real, entretanto, teria um desempenho melhor pois contaria com a presença de outros clientes ocupando o servidor, tráfego externo atrasando os pacotes do cliente, atraso nas respostas do servidor (devido a um maior número de nós entre cliente e servidor) e maior gasto computacional do servidor devido a processos internos (no teste, o servidor apenas responde requisições, não possuindo nenhum sistema *web* implementado à parte).

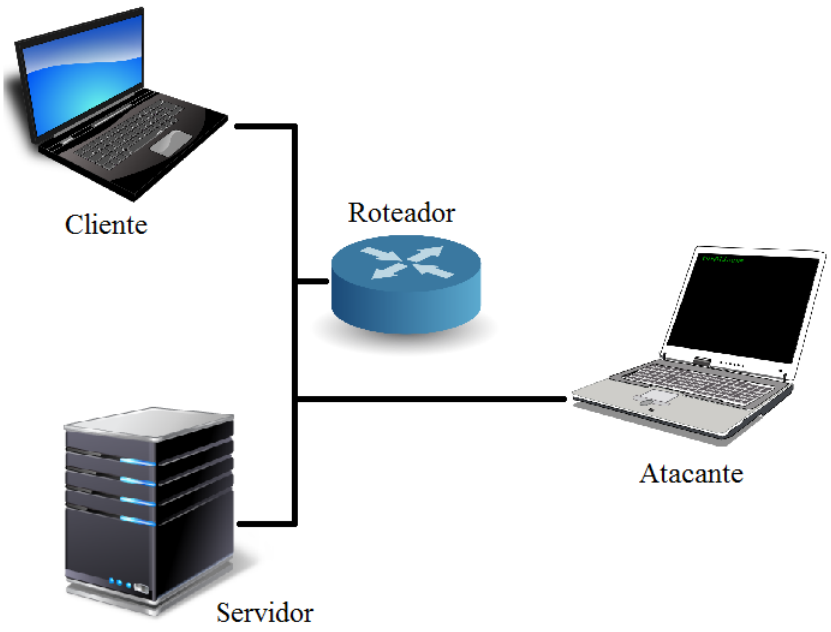

Figura 6.3: Imagem descrevendo a organização da bancada de testes. Atacante, cliente(simulando usuário legítimo) e servidor são todos ligados por cabos a um roteador exclusivo e isolado.

Segue abaixo a descrição das máquinas usadas como servidor e atacante para mérito de comparação.

Tabela 6.2: Configuração de hardware e software das máquinas atacante e servidor.

ATACANTE	SERVIDOR
Notebook Samsung®	Desktop customizado
Kubuntu(Ubuntu 13.10)	Kubuntu(Ubuntu 13.10)
Intel®Core i7 2.2GHz	Intel®Core i7 2.93GHz
8GB memória RAM	8GB memória RAM
	Apache Server 2.4.6(Ubuntu)®

Pode-se ver da tabela 6.2 que atacante e servidor possuem configurações bem parecidas, com o servidor possuindo um processador um pouco melhor.

Para a análise do tráfego durante o ataque foi usada a ferramenta Wireshark™ [12] que é capaz de captar todos os pacotes de rede em um nó e tem diversas funções de análise. Todas as três máquinas usadas para teste tiveram seu tráfego monitorado com a ferramenta. Para monitorar o estado do servidor, foi usado uma ferramenta própria do servidor Apache™ [11]. A ferramenta comumente chamada de *server-status* é disponibilizada junto com a instalação do servidor Apache™usado. O *server-status* assim que requisitado gera uma página html com informações sobre o estado do servidor, entre essas informações tem-se o número de conexões do servidor e seus estados (ocupadas, ociosas, finalizadas e etc), o consumo de CPU, as últimas requisições feitas ao servidor, lista de clientes, tempo do servidor e várias estatísticas. A ferramenta por ser bem completa e de fácil manuseio foi usada para monitorar os efeitos do ataque no servidor.

O método de testes usado foi bem simples e intuitivo. Para testar as ferramentas de ataque foi primeiramente preparada a rede, como já citado anteriormente. Com a rede montada, as três máquinas tiveram a ferramenta Wireshark™iniciada. Uma das ferramentas de ataque é, então, executada no computador atacante. Como a natureza do ataque não é muito constante o seu efeito foi medido ao longo do tempo de execução, foi escolhido fazer medições a cada cinco minutos de ataque. Como medições posteriores aos 15 minutos de ataque não mostraram muitas ocilações, foi escolhido fazer medições aos cinco, dez e quinze minutos.

A cada medição(aos 5, 10 e 15 minutos) foi usado a ferramenta *server-status* no servidor para se obter informações sobre sua degradação, a ferramenta Wireshark™também foi usada para saber o acrécimo de tempo que o cliente sofreu em suas requisições[3]. Essas requisições serviram para medir qual o nível de degradação de serviço que o cliente sofreu, foram feitas por volta de cinco requisições, o tempo médio foi tomado e os valores foram comparados com valores do servidor em condições normais.

As medições foram feitas em todas as ferramentas de ataque citadas, uma a uma, usando-se diferentes números de instâncias(no caso da API libevent que simula vários clientes em uma mesma máquina) e threads. As ferramentas foram testadas em separado, e a cada troca de ferramenta de ataque o servidor foi reiniciado para garantir a limpeza de possíveis resquícios de testes anteriores.

[3]Foram feitas requisições de páginas simples e leves para esse teste, como o ataque funciona em servidores com TLS/SSL foram feitas requisições do mesmo caráter. Como o fluxo é cifrado, as medições de tempo foram feitas do início do *handshake* TLS/SSL até seu final, pois a criptografia não possibilita saber qual o conteúdo das requisições(pacotes cifrados). Por consequência dessa medição, o tempo do atraso real sofrido pelo cliente é de fato maior que o registrado.

Capítulo 7

Resutados

Esse capítulo mostra os resultados obtidos com os testes descritos no capítulo anterior. Os resultados serão mostrados graficamente para uma melhor compreensão e as ferramentas serão comparadas e comentadas, desenvolvendo uma melhor noção do potencial do ataque.

7.1 Primeira versão da ferramenta

As ferramentas foram brevemente descritas no capítulo anterior, para uma breve comparação pode-se usar a tabela 6.1 como referência. A primeira versão da ferramenta foi confeccionada usando o *framework*/API QT®. A ferramenta segue a estrutura geral similar as outras três ferramentas descritas, porém respeita a ordem de envio de pacotes (figura 6.2). Os *handshakes* feito por essa versão são feitos de forma natural, a ferramenta começa o processo de *handshake* e espera o servidor responder. O servidor responde e só então a ferramenta continua. Essa característica deixa a ferramenta ociosa enquanto espera respostas do servidor, esse breve período ocioso significa que o servidor está ocupado processando a última requisição da ferramenta. Esse período é justamente o intuito da ferramenta, gastar os recursos do servidor.

Seguir o *design* do protocolo é o esperado de qualquer cliente, logo a ferramenta possui menor chance de ser descoberta como maliciosa. Esse tempo ocioso, porém, indica que o atacante não está usando todo seu poder para incapacitar o servidor; há uma leve dilema entre uma ferramenta mais discreta ou mais eficiente.

Para servir de referência, foi medido o tempo de resposta do servidor préviamente ao ataque. O servidor normalmente demora cerca de 1.964 milisegundos para responder a uma requisição, possui cerca de 50 conexões disponíveis de um total de 50 conexões e tem um consumo de CPU em cerca de 0.09%. Essas medições foram feitas com as ferramentas descritas no capítulo anterior. As figuras 7.1 e 7.2 mostram, respectivamente, o tempo que as requisições começaram a tomar e a relação proporcional do aumento. Esse aumento proporcional é dado pela razão entre o novo tempo e o tempo pré-ataque, dando uma medida de quantas vezes o ataque aumentou o tempo de resposta.

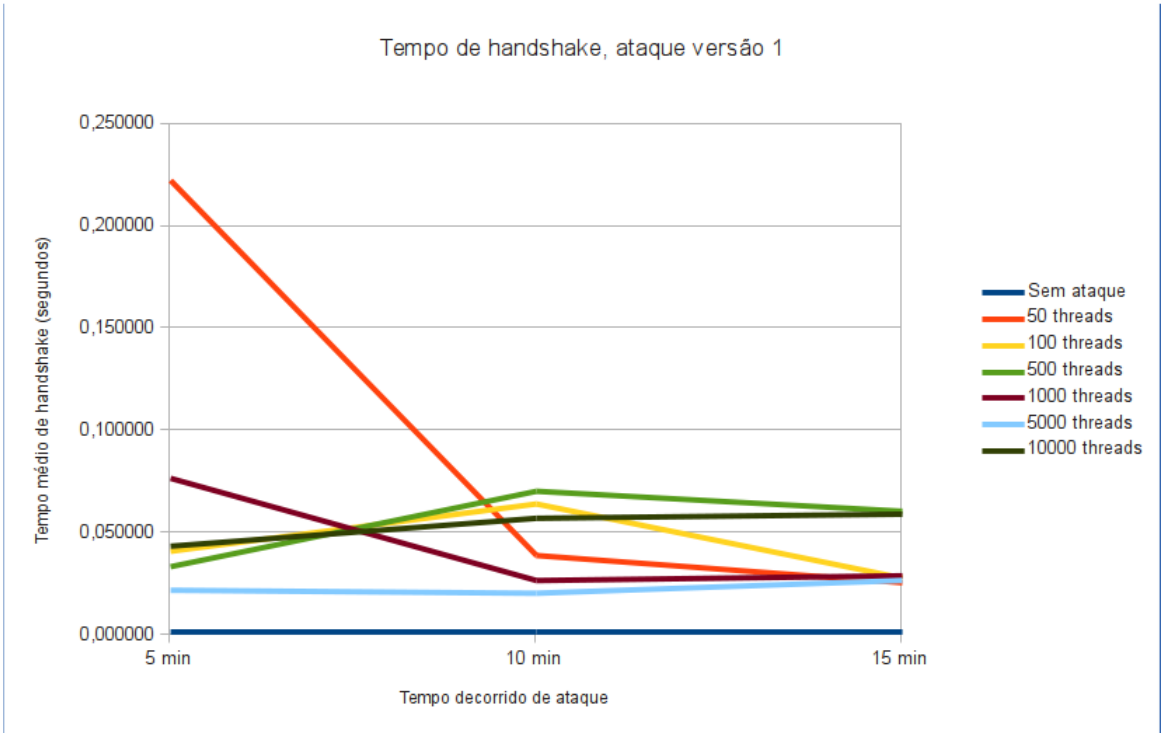

Figura 7.1: Gráfico com os tempos médios de duração de handshake durante o ataque com a primeira versão da ferramenta.

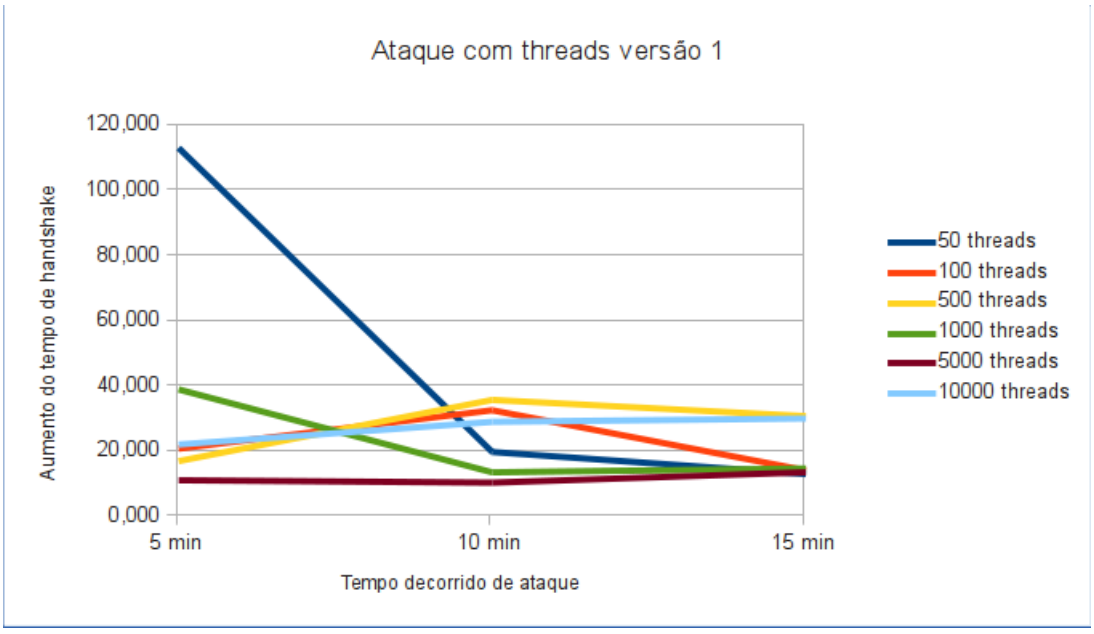

Figura 7.2: Gráfico com o aumento proporcional do tempo de resposta gerado pela primeira versão da ferramenta.

Pode-se ver na figura 7.2 o tempo decorrido de ataque no eixo horizontal, e no eixo vertical o aumento do tempo de resposta de cada requisição. O gráfico mostra que ao executar a ferramenta usando 50 *threads* o servidor não está preparado e acaba por ter um

grande aumento no tempo de resposta, quase que 120 vezes maior que o tempo normal de resposta. Porém, servidores como o Apache®são dinâmicos, eles usam alocação dinâmica para aumentar recursos em tempo real e garantir sua qualidade de serviço. Olhando o funcionamento do ataque percebe-se que o mesmo demostrou-se bem ocilante, porém o aumento do tempo de resposta é real, testes com um maior número de *threads* mostraram que qualquer requisição feita por um cliente, por mais simples que seja, têm seu tempo aumentado entre 20 e 40 vezes. Esse aumento pode afetar clientes usando de sistemas que usem de aplicações envolvendo tempo real, porém mesmo com o aumento o servidor continua ativo e respondendo, logo a ferramenta se encaixaria como negação de serviço parcial.

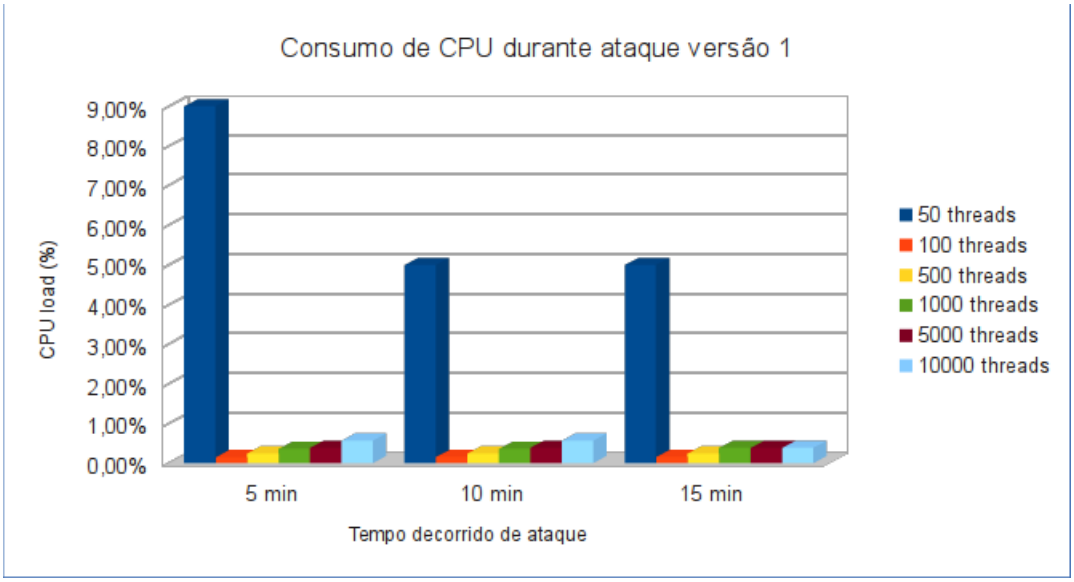

Figura 7.3: Gráfico com o consumo de CPU gerado pela primeira versão da ferramenta.

O gráfico da figura 7.3 mostra que o ataque aumenta o consumo de CPU de 0.09% para valores um pouco maiores, cerca de 0.5%. O aumento moustrou-se pouco, não sendo muito relevante para uma negação de serviço. Valores tão baixos no consumo de CPU não chegam a explicar o aumento relativo no tempo de resposta, provavelmente o consumo é intermitente tendo seu pico apenas em momentos onde o servidor tem que decifrar o pacote com 'lixo' e a ferramenta de medição acaba por não captar esses picos e mostra apenas o consumo estável do CPU. Uma outra possibilidade, é a de que o caráter levemente volumétrico da ferramenta(as várias *threads* atacando constantemente) esteja causando parte do atraso no tempo de resposta. Ainda olhando o gráfico, pode-se reparar que o consumo durante o teste com 50 *threads* foi bem superior, assim como a medição do tempo de resposta. Essa correlação mostra que o aumento significativo do tempo de resposta está intimamente ligada ao gasto de CPU.

Tabela 7.1: Estado das conexões ocupadas durante ataque com a ferramenta versão 1.

Conexões ocupadas (ferramenta versão 1)			
	Tempo decorrido		
Número de *threads*	5 minutos	10 minutos	15 minutos
50 threads	41/50	39/50	40/50
100 threads	49/50	49/50	49/50
500 threads	41/50	41/50	41/50
1000 threads	41/50	41/50	42/50
5000 threads	42/50	42/50	42/50
10000 threads	42/50	42/50	41/50

A tabela 7.1 mostra as conexões ocupadas e o número máximo de conexões disponíveis [1], como dito préviamente o servidor usado é dinâmico e por isso esse número de conexões pode aumentar se for necessário, porém consome mais recursos. Observando a tabela pode-se observar que grande parte das conexões fica ocupada, mas é seguido o limite de 50 conexões no servidor que não é considerado um limite alto. Um maior número de conexões implica em um maior gasto de memória por parte do servidor, o que não é o foco desse tipo de ataque, porém um maior número de conexões indica que os clientes conectados ao servidor terão que juntos 'dividir' o gasto de CPU inteiro do servidor (seguindo uma política de escalonamento definida).

A ferramenta mostrou um certo impacto no servidor, mas nada muito relevante. O servidor conseguiu se adaptar bem ao ataque deixando o cliente com prejuízos mínimos, relevante apenas para alguns tipos de aplicações.

7.2 Segunda versão da ferramenta

A segunda versão da ferramenta tenta maximizar o gasto de CPU do servidor tentando mantê-lo sempre ocupado, para isso sempre que possível a ferramenta manda as requisições do *handshake*. A ferramenta manda primeiramente o *client hello* (para mais detalhes rever o capítulo sobre TLS/SSL) e logo em seguida e ininterruptamente manda o *client key exchange*, *change cipher spec* e *encrypted handshake message*. O servidor então considera que o cliente foi muito rápido e começa a processar as respostas, ou pode, também, ocorrer em alguns casos do servidor considerar a mensagem como errada e encerrar a conexão. Em caso de falhas, ou da conexão ser encerrada, a ferramenta restabelece a conexão e começa o processo novamente. A idéia é verificar a conexão, caso esteja ativa despejar o mais rápido possível os *handshakes* sem permitir descanso ao servidor. Em todos os outros méritos a ferramenta segue o fluxograma da figura 6.2.

Os gráficos das figuras abaixo mostram o aumento do tempo de resposta do servidor em relação a antes do ataque.

[1] A tabela mostra o número de conexões efetivamente ocupadas, por maior que seja o número de *threads* muitas das conexões já foram encerradas ou estão sendo encerradas, logo o número de conexões ocupadas não alcançam as conexões máximas. Além disso, servidores *web* usam de processos de escalonamento e filas para receber conexões de novos clientes.

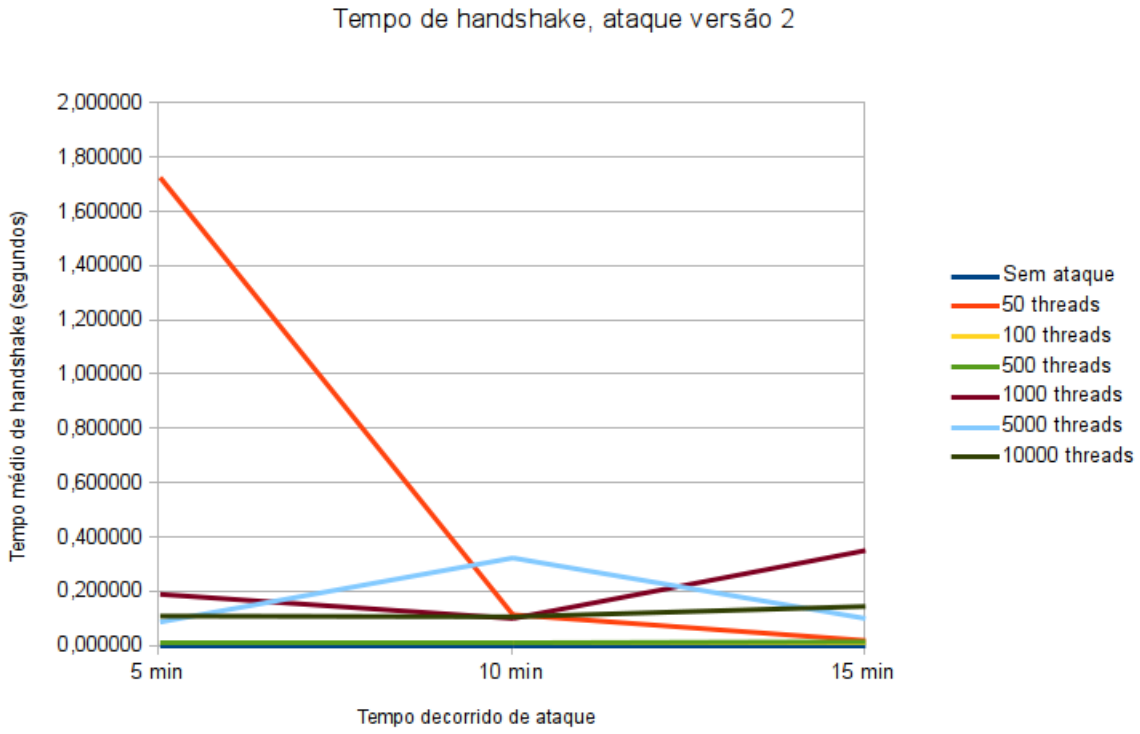

Figura 7.4: Gráfico com os tempos médios de duração de handshake durante o ataque com a segunda versão da ferramenta.

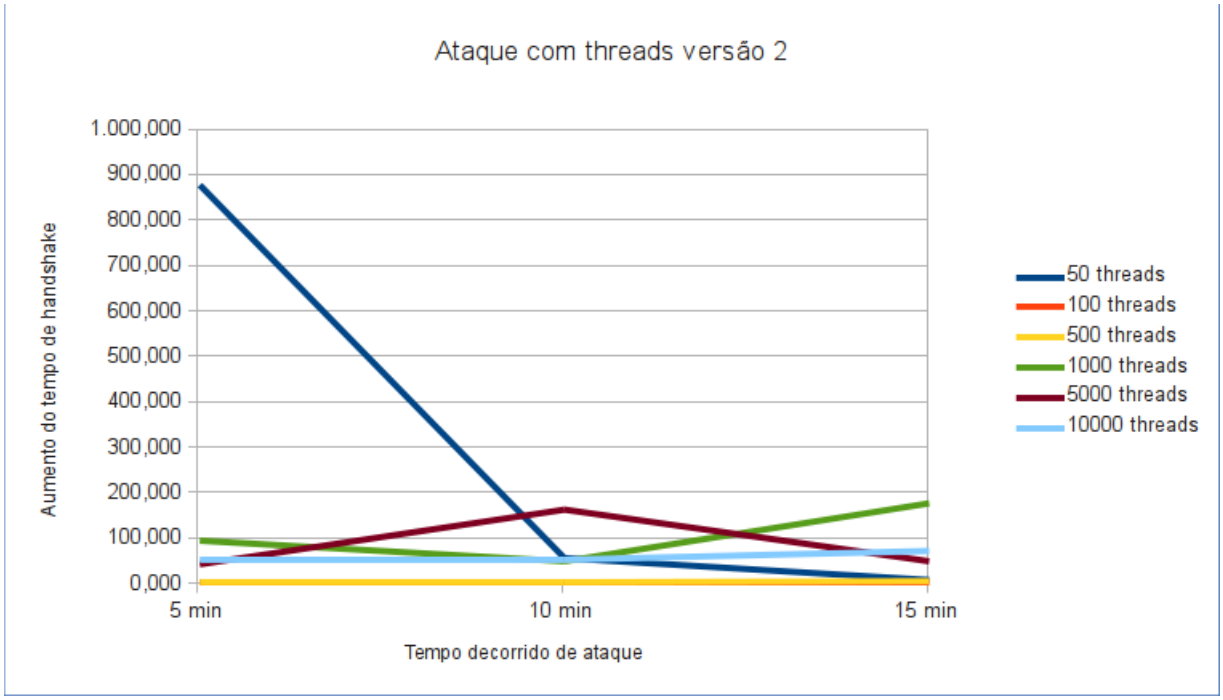

Figura 7.5: Gráfico com o aumento proporcional do tempo de resposta gerado pela segunda versão da ferramenta.

Nos gráficos 7.5 7.4 observa-se que novamente a ferramenta rodando com 50 *threads* teve um primeiro resultado bem elevado e depois abaixa ficando bem próxima dos testes com maior número de *threads*. Nessa versão, entretanto, o acrécimo é bem maior, a versão com 50 *threads* chegou a atrasar as requisições do cliente no seu pico em quase 900 vezes. Os testes com número diferentes de *threads* ficaram mais estáveis, dando-se destaque as versões com 1000 e 5000 *threads*. Esses testes tiveram um bom desempenho quase alcançando um aumento de 200 vezes. Esse aumento começa a ser significativo para o ataque, uma requisição simples de uma página que antes demorava alguns milisegundos agora demora segundos.

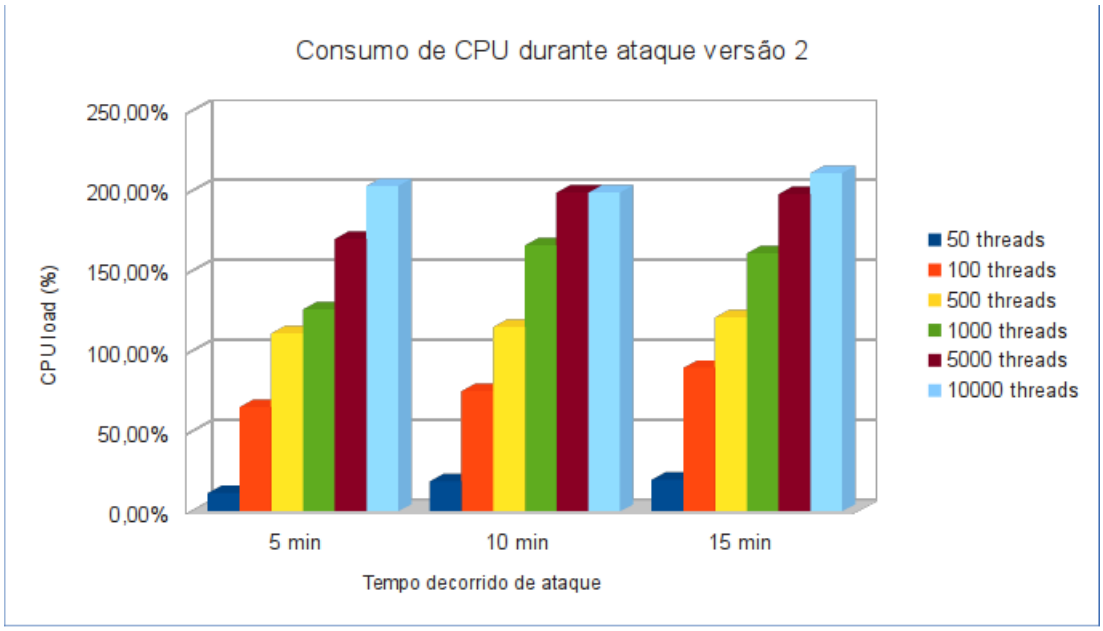

Figura 7.6: Gráfico com o consumo de CPU gerado pela segunda versão da ferramenta.

A figura 7.6 do gráfico acima mostra o consumo de CPU que também aumentou em relação a primeira versão da ferramenta, as medições mostraram que aumento do gasto de CPU foi proporcional ao aumento do número de *threads*. O consumo máximo obtido com a primeira versão foi 9%, com essa versão obtêm se um percentual maior que 200% do consumo[2]. O elevado consumo de CPU mostra que ataques do genêro são realmente viáveis.

Valores de consumo de CPU acima de 100% se dão devido ao fato do servidor alocar dinamicamente novos processos para conseguir suportar a demanda exigida, o que acaba por usar de mais de um núcleo do processador.

[2] Valores acima de 100% são devido ao consumo de CPU maior do que o máximo reservado ao uso do servidor, porém com comportamento dinâmico e uso de multiplos *cores* permite um consumo maior do que a medida de 100%.

Tabela 7.2: Estado das conexões ocupadas durante ataque com a ferramenta versão 2.

Conexões ocupadas (ferramenta versão 2)			
	Tempo decorrido		
Número de *threads*	5 minutos	10 minutos	15 minutos
50 threads	42/50	41/50	42/50
100 threads	42/50	43/50	42/50
500 threads	41/50	42/50	42/50
1000 threads	41/50	42/50	42/50
5000 threads	41/50	42/50	42/50
10000 threads	42/50	41/50	41/50

A tabela mostra que o número de conexões máximas não chegou a aumentar e o número de conexões ocupadas parece permanecer o mesmo. Não se pode dizer muito das conexões, porém o aumento de CPU e do tempo de resposta ainda mostraram que o ataque pode ser bem efetivo.

7.3 Versão da ferramenta com libevent e RSA

As ferramentas que usam da API libevent obtiveram um maior desempenho. A API é especializada em chamadas e controles de evento do sistema operacional, como o uso de *sockets* e de funções de rede, além disso ela é especializada no tratamento de funções por evento e no uso de processos simultâneos [19]. Dados essas especialidades da API o casamento com a ferramenta foi bem efetivo, pois a grosso modo a ferramenta não passa de várias funções de rede(chamadas de sistema) executando simultaneamente e o mais rápido possível. A ferramenta e a API funcionam baseadas em eventos, assim que a ferramenta detecta um evento ela já dispara o próximo evento escolhido. Assim que uma mensagem do servidor é recebida a resposta já está na iminência de ser enviada(próximo evento). Além disso, execuções simultâneas usando *threads* tem um custo elevado de CPU e memória, a API escolhe a melhor e mais rápida opção de escalonamento para realizar eventos simultâneos o mais rápido possível, possuindo menos custo computacional que as *threads*. Esta versão da ferramenta usa apenas de *cipher suites* com RSA, justamente para se ter a comparação com o *forward secrecy*, a ferramenta como já mencionado também segue o protocolo a risco, com mensagens alternadas entre cliente e servidor sendo respeitadas.

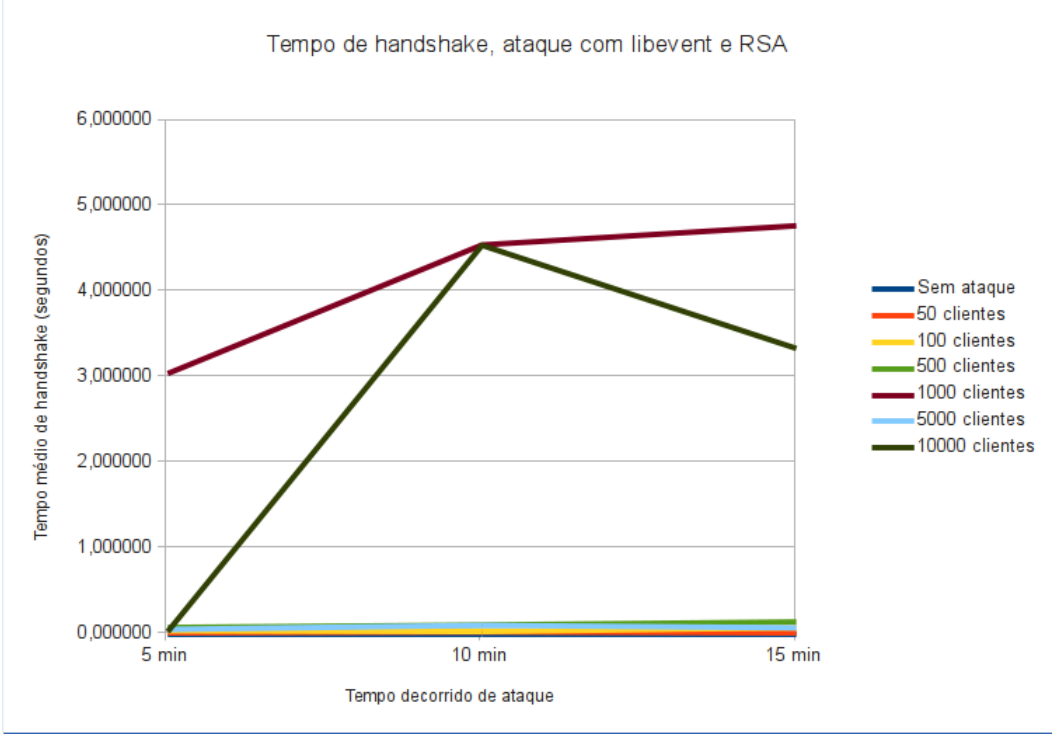

Figura 7.7: Gráfico com os tempos médios de duração de handshake durante o ataque com a versão da ferramenta usando libevent e RSA.

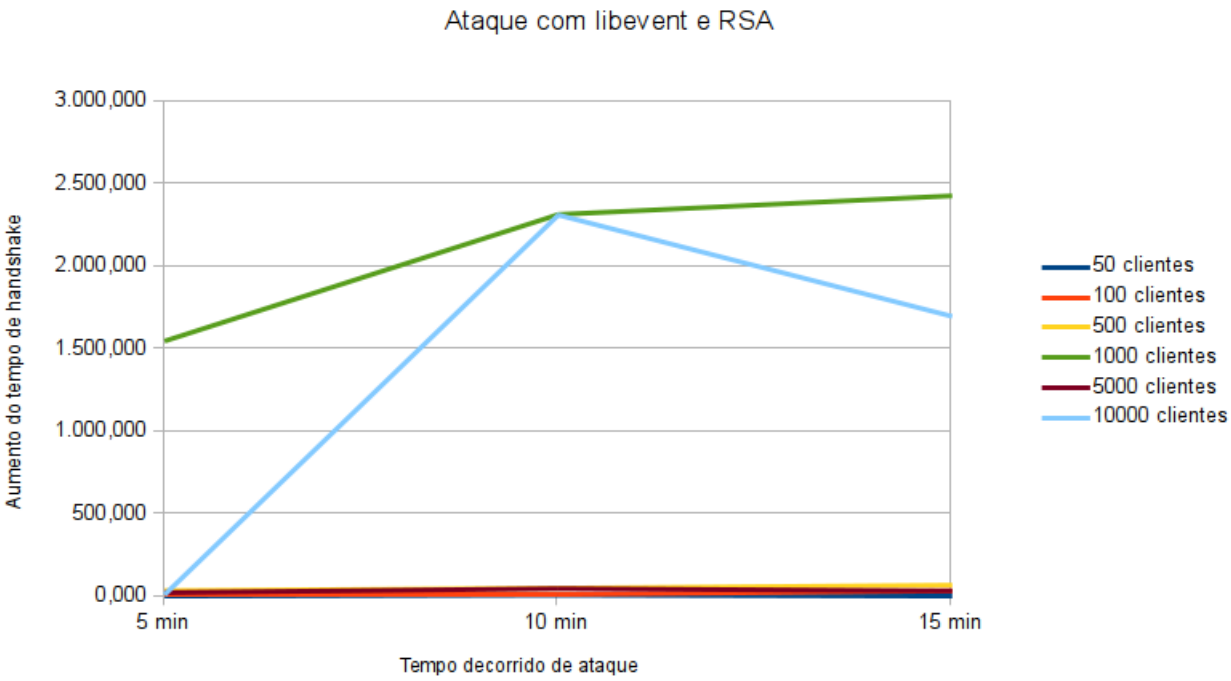

Figura 7.8: Gráfico com o aumento proporcional do tempo de resposta gerado pela versão da ferramenta com libevent e RSA. Nota-se o termo 'clientes' para se referir as múltiplas instâncias usadas pela API.

As figuras 7.8 7.7 mostram que a versão do ataque conseguiu ser mais eficiente que todas as outras ferramentas testadas até agora, as execuções com 1000 e 10000 clientes/instâncias foram bem melhores e chegaram quase a um aumento de 2500 vezes no tempo normal para o *handshake*. As outras execuções entretando ficaram com um desempenho menor do que 200 vezes. Essa diferença discrepante ao se variar o número de clientes acaba por mostrar que o número de clientes para o ataque deve ser calculado previamente, o consumo de memória da máquina atacante provavelmente é o responsável por essa flutuação no gráfico, porém o aumento do tempo de resposta ainda é aceitável e comparável as ferramentas usando *threads*.

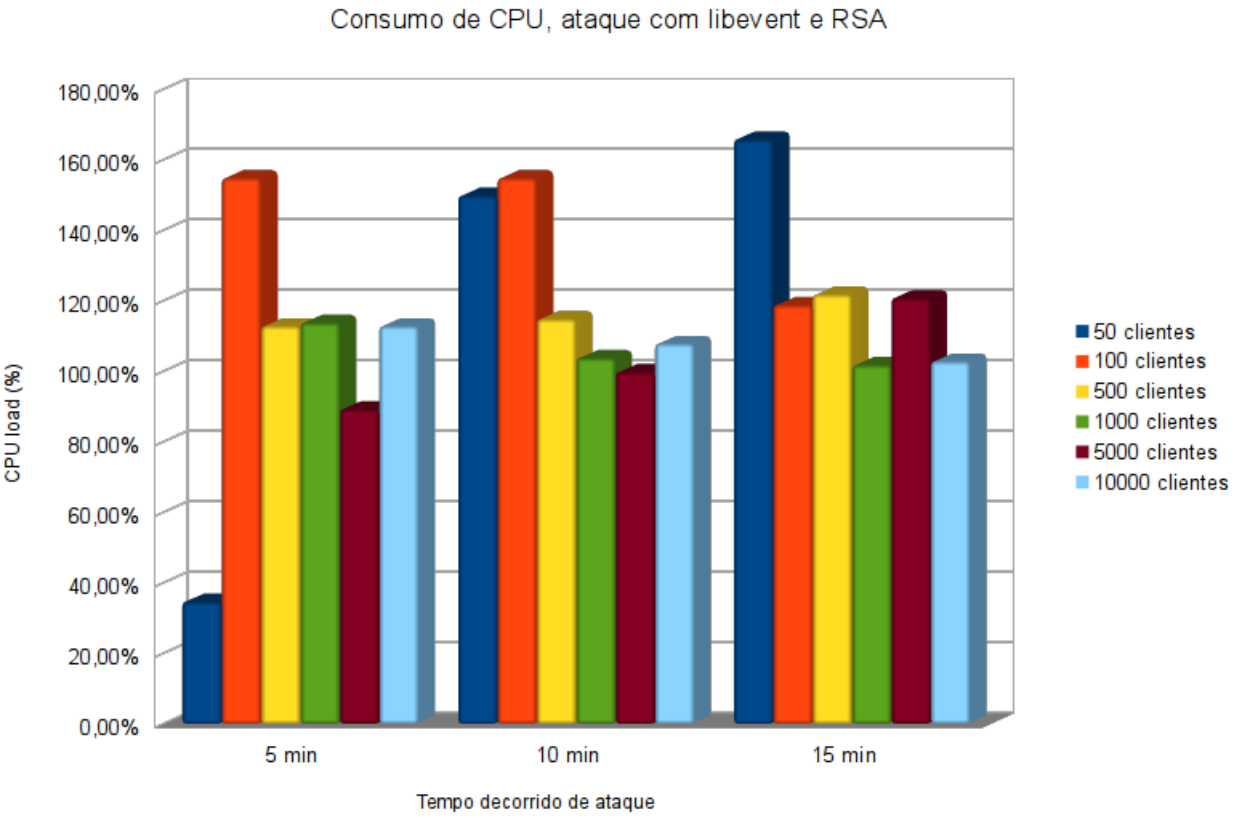

Figura 7.9: Gráfico com o consumo de CPU gerado pela versão da ferramenta com libevent e RSA.

O consumo de CPU mostrado na figura 7.9 mostra que quase todas as medições feitas forçaram o CPU a um consumo maior que 100%, o que é bem preocupante para o servidor. Mas mesmo com esse alto consumo, os maiores valores alcançados foram entre 140% e 160%, o que representa números menores que na segunda versão da ferramenta usando threads. A demora no tempo de resposta está ligada ao consumo excessivo de CPU por parte do servidor, entretanto essa ferramenta possuiu um maior tempo de resposta e um menor gasto de CPU em relação a segunda versão da ferramenta. Essa diferença está bem explicada pelo número de conexões, como a API libevent permite maior rapidez nas funções de rede e maior processamento paralelo, o número de clientes ativos e efetivamente ocupando conexões é maior. Mais detalhes podem ser vistos na tabela 7.3 abaixo.

Tabela 7.3: Estado das conexões ocupadas durante ataque com a ferramenta libevent e RSA.

Conexões ocupadas (ferramenta com libevent e RSA)			
	Tempo decorrido		
Número de *instâncias/clientes*	5 minutos	10 minutos	15 minutos
50 clientes	46/75	48/75	48/75
100 clientes	62/100	62/100	67/100
500 clientes	142/150	132/150	136/150
1000 clientes	143/150	129/150	142/150
5000 clientes	147/150	129/150	144/150
10000 clientes	146/150	108/150	133/150

Como mostrado na tabela 7.3, o número de conexões máximas do servidor aumentou dinâmicamente, originalmente o servidor começa com 50 conexões disponíveis, nos testes com a ferramenta usando a libevent o número de conexões começou direto em 75 pois mesmo com apenas 50 clientes o servidor não estava conseguindo gerenciar tantas conexões. Posteriormente o número de conexões aumenta para 75 e depois para 150, onde permanece estabilizado. Esse maior número de conexões máximas indica que o atacante estava conseguindo efetuar mais conexões devido a maior agilidade no seu mecanismo de paralelismo e envio de pacotes, pois o mesmo número de clientes/intâncias confere com o mesmo número de *threads* das outras ferramentas. Esse aumento se dá exclusivamente ao uso da API libevent.

7.4 Versão da ferramenta com libevent e Diffie & Hellman

A última ferramenta usa da API libevent e de *cipher suites* usando o protocolo de troca de chaves Diffie & Hellman efêmero para colocar à prova se o *perfect foward secrecy* traz consigo consequências negativas. A ferramenta foi feita no mesmo molde da ferramenta anterior (RSA) e respeitando a troca de mensagens intercaladas entre cliente-servidor, porém usa apenas de *perfect foward secrecy*.

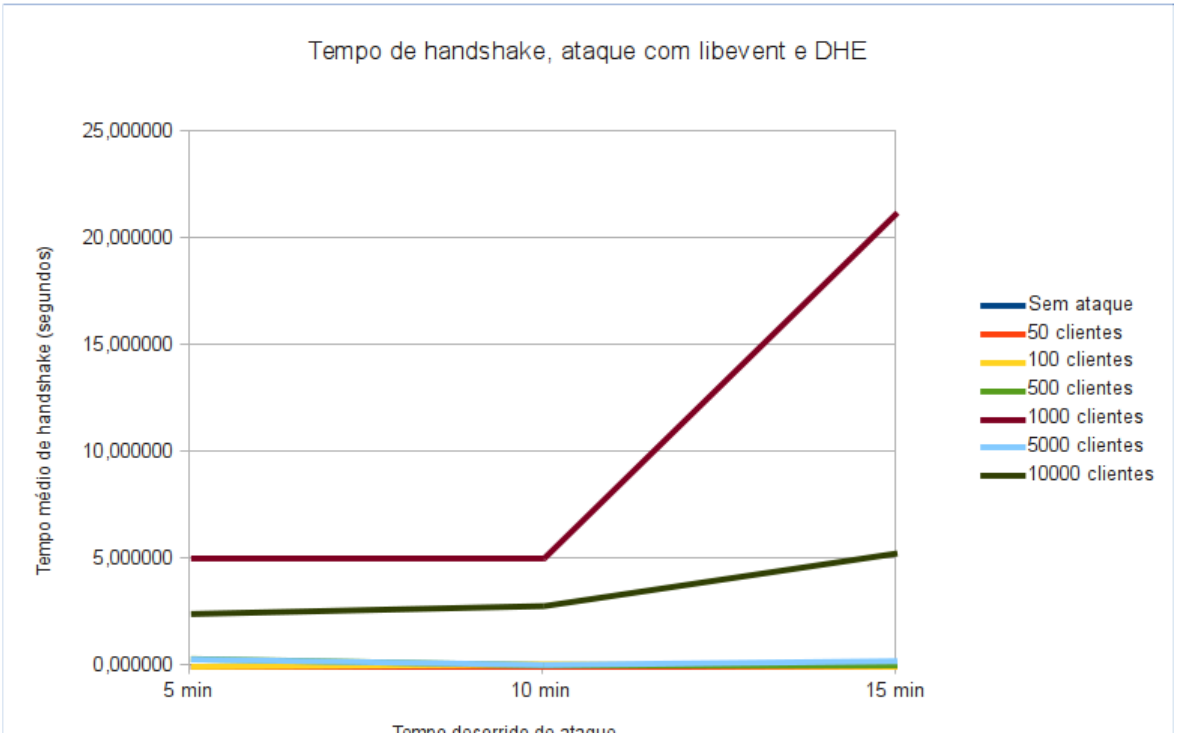

Figura 7.10: Gráfico com os tempos médios de duração de handshake durante o ataque com a versão da ferramenta usando libevent e Diffie & Hellman(*Perfect Foward Secrecy*).

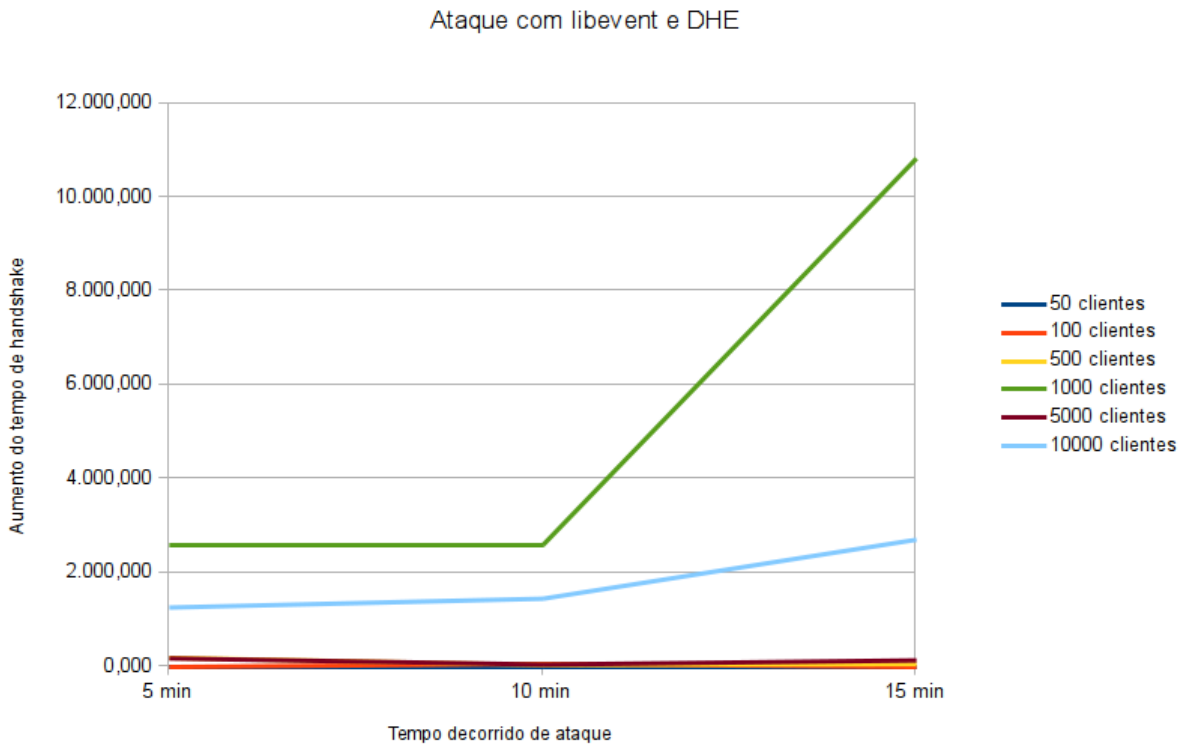

Figura 7.11: Gráfico com o aumento proporcional do tempo de resposta gerado pela versão da ferramenta com libevent e Diffie & Hellman.

Na figura 7.11 pode-se observar que os valores estão bem mais altos que comparado a outras ferramentas. A medição feita com 1000 clientes/instâncias chega a aumentar o tempo de resposta em 10900 vezes, o que dá um bom tempo de espera para o cliente, ainda mais se tratando de uma simples requisição de uma página de poucos bytes. As outras medições também tiveram bons resultados, a medição com 10000 clientes ficou com um acrécimo sempre acima de 1000 vezes o valor original, já as outras medições tiveram valores similares as ferramentas usando *threads*.

A grande variação entre as diferentes medições ressalta que esse ataque necessita de uma calibração de parâmetros, pois o ataque possui muitas ocilações e diferentes números de clientes/instâncias fazem toda a diferença, como pode-se observar nos gráficos já mostrados. A escolha de testar os ataques para diferentes números de clientes e *threads* é justamente para prever em quais faixas de parâmetros a máquina atacante consegue causar mais impacto ao servidor. Realizar novos testes na ferramenta com os parâmetros em faixas de valores próximos aos melhores resultados pode inclusive revelar melhores resultados e mais dados sobre o ataque, porém testes exaustivos para alcançar melhores resultados fogem do escopo deste documento. Este documento preza apenas em alertar sobre a possibilidade de um ataque e mostrá-lo à comunidade, desenvolver e evoluir constantemente o ataque está longe da sua índole.

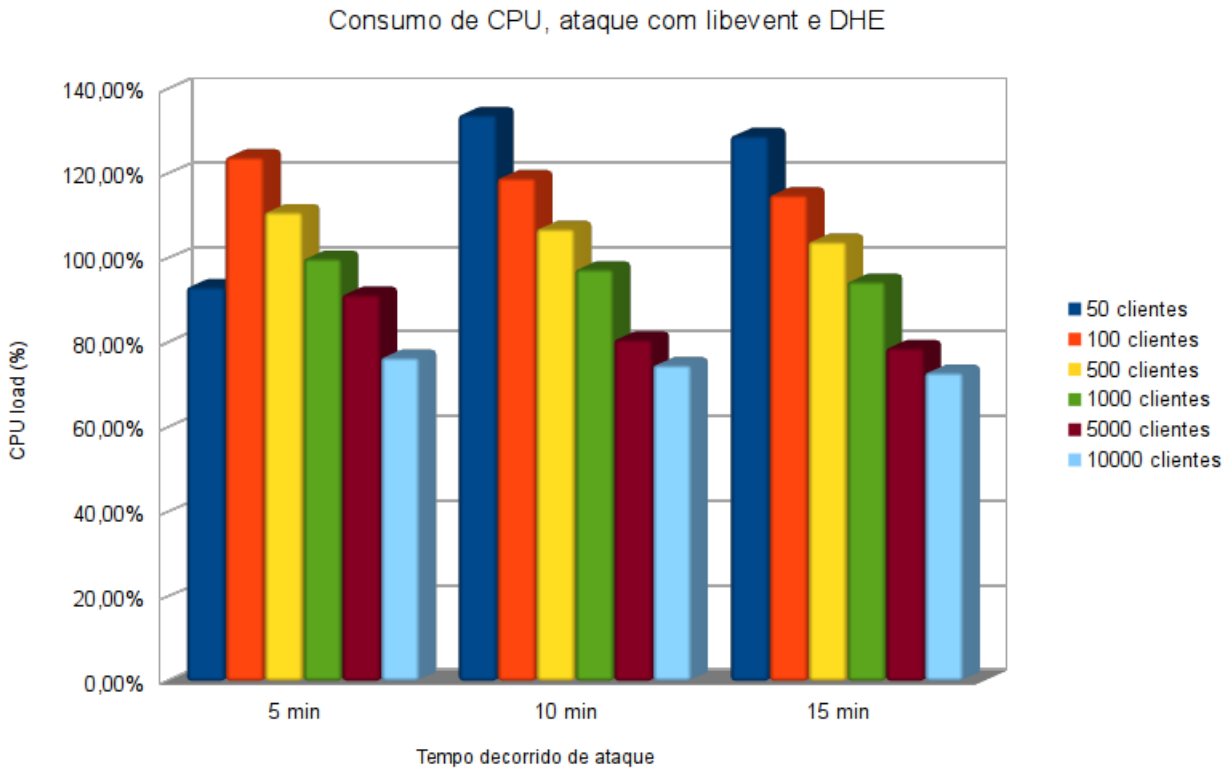

Figura 7.12: Gráfico com o consumo de CPU gerado pela versão da ferramenta com libevent e Diffie & Hellman.

O consumo de CPU da ferramenta possui os mesmos moldes da ferramenta usando de RSA, como pode-se averiguar na figura 7.12 acima. Entretanto, o consumo usando a ferramenta tem um comportamento decrescente, quanto menor o número de clientes maior o consumo de CPU e quanto maior o número de clientes menor o consumo. Ao se observar a figura 7.11(referente aos tempos de resposta), os maiores aumentos nas respostas são de testes com maior número de clientes, 1000 e 10000 clientes.

Observando a ferramenta com a API libevent e RSA, pode-se reparar um leve tendência decrescente no consumo de CPU, porém o padrão é menos claro que no gráfico acima. Comparando os padrões nas ferramentas que usam *threads* e QT®, nota-se que o padrão é inverso, possuindo comportamento crescente. Tal comportamento se deve ao maior sucesso das ferramentas usando a API libevent, pois com maior número de conexões e maior gasto computacional o servidor sobrecarregado tenta alocar dinamicamente mais recursos na tentativa de suportar a sua demanda.

Tabela 7.4: Estado das conexões ocupadas durante ataque com a ferramenta libevent e Diffie & Hellman.

Conexões ocupadas (ferramenta com libevent e DHE)			
	Tempo decorrido		
Número de *instâncias/clientes*	5 minutos	10 minutos	15 minutos
50 clientes	46/100	42/100	48/100
100 clientes	96/150	95/150	94/150
500 clientes	144/150	136/150	126/150
1000 clientes	134/150	143/150	144/150
5000 clientes	143/150	140/150	143/150
10000 clientes	141/150	124/150	125/150

Na tabela 7.4, que descreve a ferramenta usando de Diffie & Hellman, o número máximo de conexões começa em 100, mas por volta de apenas metade permanece ocupada. Em seguida, o número de conexões aumenta consideravelmente chegando no limite de 150 conexões, com 100 clientes por volta de 90 conexões permanecem ocupadas. A partir de 500 clientes quase todas as conexões permanecem ocupadas, ficando acima de 120 e perto de 150 conexões máximas. Nota-se em casos anteriores, que com um número de conexões disponíveis tão baixas o servidor já teria aumentado o número de conexões máximas, caracterizando um comportamento diferente.

No geral, as ferramentas usando da API libevent causaram um maior atraso para o cliente(usuário legítimo), que é a medida mais confiável da negação de serviço sofrida. A segunda versão da ferramenta usando de QT®, entretanto, conseguiu um maior consumo de CPU mas um menor número de conexões. Esse gasto de CPU elevado em relação as ferramentas com libevent mostrou que existe uma correlação entre consumo de CPU e número de conexões efetivamente ocupadas, e que essa relação não é linear. As ferramentas usando libevent obtiveram um maior tempo de resposta mesmo com um consumo de CPU um pouco menor que a segunda versão da ferramenta com QT, mostrando que o aumento dos dois parâmetros medidos causa impacto, mas não necessariamente o aumento de apenas um deles é suficiente para dizer o impacto do ataque.

O ataque em geral se mostrou bem oscilante com o decorrer do tempo. O aumento do tempo de resposta do servidor variou ao longo dos ataques e não mostrou nenhum padrão aparente, porém seu impacto foi real. O ataque possui um comportamento de difícil previsão, mas está sempre exaurindo o servidor e por mais que seja difícil prevê-lo os resultados obtidos nas versões usando a API libevent foram todos altos.

A versão do ataque usando de libevent e *Perfect Foward Secrecy* mostrou-se a mais eficiente das ferramentas e consequentemente a mais perigosa. Mais detalhes e conclusões sobre os testes realizados serão abordados no próximo capítulo.

Capítulo 8

Conclusões

8.1 Análise e classificação

Como os resultados mostraram, as ferramentas de ataque desenvolvidas nesse documento conseguem atrasar bastante o tempo de resposta de um servidor pequeno mas bem equipado, consequentemente degradando o serviço para o cliente. Apesar do servidor não ter sofrido uma negação de serviço total, o ataque é viável e pode ser melhorado em alguns aspectos. O ataque pode evoluir rapidamente se tornando uma ameaça e cabe aos estudioso da área de segurança de redes planejar métodos de minimizar os efeitos do ataque antes que o mesmo se torne popular.

Os resultados obtidos foram todos realizados em ambiente controlado e isolado para uma medição mais precisa, mas, em ambiente real, estipula-se que o ataque seja bem mais eficiente. Em ambiente real, há vários nós entre cliente e servidor, logo há um maior tempo de resposta natural. Um servidor real quase sempre possue uma aplicação ou um sistema web em execução, que consome mais recursos do servidor facilitando a negação de serviço. Além disso, há tráfego externo e outros clientes que irão ocupar banda, atrasar pacotes e que também irão consumir mais recursos do servidor. Esses acrécimos podem ser a diferença entre uma negação de serviço total e parcial, confirmando a viabilidade de ataques do gênero.

Como parte da análise do ataque e para um melhor entendimento é importante classificá-lo. As ferramentas de ataque assim como apresentadas podem ser classificadas como um ataque de negação de serviço (DoS) degradante que causa uma negação parcial de serviços. O ataque também é considerado como um ataque de conteúdo que visa o uso previsto do protocolo para consumir o CPU da vítima. A vítima do ataque sofre oscilações na negação de serviço, não possuindo nenhum caráter constante, porém o ataque é caracterizado por ter sempre um fluxo constante.

8.2 Sugestões para a evolução do ataque

Observando os resultados, fica claro que as ferramentas de ataque obtiveram resultados diferentes e algumas ferramentas se destacaram das demais. Essas ferramentas podem ser trabalhadas, por exemplo, para se montar um ataque de negação de serviço distribuído(DDoS) aumentando drasticamente o poder do ataque. As ferramentas chegaram a

obter resultados capazes de atrapalhar qualquer cliente, usando de algumas modificações e incrementos as ferramentas podem, com certeza, realizar uma negação de serviço total e disruptiva.

Para melhorar as defesas contra o ataque é necessário levar em conta a evolução do ataque e para onde ele caminha, para evitar ser pego desprevinido. Comparando as ferramentas que usavam *threads* e QT®fica claro que uma abordagem menos discreta e mais rápida possue melhores resultados, olhando as ferramentas que usam libevent percebe-se um maior ganho no aumento do número de conexões e no tempo de resposta. Juntando o melhor das duas ferramentas supõe-se que criar uma ferramenta usando as funções de sistema e paralelismo da API libevent junto com a abordagem da ferramenta que não espera respostas do servidor pode criar uma ferramenta de ataque mais poderosa. Essa ferramenta precisaria de algumas modificações na API libevent para garantir que a mesma não esperasse eventos(resposta do servidor) ou que usasse um evento de *timer* bem curto. Executar o ataque de mais de uma máquina assim como os DDoS também pode aumentar o perigo do ataque.

8.3 Possíveis contramedidas

O ataque não pode ser diretamente evitado, pois o ataque usa corretamente do protocolo TLS/SSL e uma vez que o pacote com 'lixo' é decifrado, o gasto de CPU do servidor já ocorreu e evitar o pacote errado não é mais viável. Um *handshake* TLS/SSL normal pode falhar naturalmente o que deixa impossível de se prever se o pacote com 'lixo' foi intencional. Entretanto, o ataque pode ser detectado e se detectado, pode ser evitado com o uso de uma *firewall* usando uma *web application* que registre estatísticas de falhas decorrentes de clientes.

Para detectar o ataque basta usar estatísticas do servidor para medir erros de decifração ou contar alertas TLS/SSL emitidos, compara-se em seguida essas estatísticas com os valores que normalmente são obtidos, em caso de discrepância, um ataque pode estar ocorrendo. Esses valores para comparação podem ser obtidos de forma estática, automática e semi-automática. Para mais detalhes sobre as vantagens e desvantagens na escolha de limiares de comparação, verificar o capítulo 3 e a subseção de mecanismos de defesa.

Uma solução reativa(após a detecção) proposta para impedir o ataque é o uso de uma firewall bloqueando o atacante. Como o atacante tem que estabelecer uma conexão TCP para começar o ataque, não há a possibilidade de *ip spoofing* e logo o endereço IP do atacante é revelado. Essa solução aumenta o tempo de resposta do servidor pois o mesmo começa a filtrar usuários um por um, o que em alguns casos ajuda o atacante. Essa solução, entretanto, não é perfeita, pois uma versão bem feita do ataque como DDoS(ataque distribuído com várias máquinas) pode simplesmente alternar entre as máquinas atacantes infectadas para evitar detecção ou atravessar o bloqueio da firewall. Toda solução de bloqueio, no entanto, pode acabar por bloquear tráfego legítimo devido a falsos positivos.

Uma outra solução é aumentar a capacidade do hardware do servidor para que o mesmo consiga executar suas funções mais rápidas e mais eficientes e suporte o ataque sem trazer consequências danosas aos clientes. Essa solução resolve quase todos os tipos de DoS, porém é meio utópica pois exige um alto custo financeiro e às vezes, tecnologia ainda não disponível. Em vez de melhorar o hardware do servidor, também pode-se usar aceleradores

de SSL. Esses aceleradores aumentam o desempenho de funções criptográficas, porém também possuem custo e apenas mitigam o ataque.

Um ponto importante é o uso de *cipher suites* adequadas, elas possuem tempo de respostas diferentes em cada servidor e escolher as certas podem mitigar os efeitos do ataque ou aumentá-lo. Vale a pena executar um teste de desempenho no servidor e ponderar entre desempenho e segurança, para escolher quais *cipher suites* são ideais de serem disponibilizadas aos clientes. Esse ponto remete à polêmica envolvendo a NSA [13] e a discussão sobre o uso de *perfect foward secrecy*. Analisando os resultados pode-se reparar que o uso de *perfect foward secrecy* aumenta consideravelmente o potencial do ataque, então não há muito o que se discutir, tudo se resume ao dilema entre segurança e eficiência. O uso de protocolos com Diffie & Hellman efêmero tem um alto custo computacional e considerável fragilidade de disponibililidade(em caso de ataques DoS) em troca de sua maior segurança criptográfica.

Levando em conta todos os pontos citados, pode-se dizer mais uma vez que o ataque de negação de serviço proposto é viável e pode ser usado com intenções maliciosas. As ferramentas aqui propostas podem inclusive ser melhoradas para um maior impacto, por isso são necessárias medidas para se defender do ataque. Algumas sugestões foram propostas, mas outras devem ser propostas e testadas. Encerra-se este documento com um apelo em prol da segurança de redes e sua importância no mundo moderno.

Referências

[1] J. Abbate. Inventing the internet. *Cambridge: MIT Press*, 1999. 1

[2] Blaise B. Posix threads programming. https://computing.llnl.gov/tutorials/pthreads/. [Acessado em 22/04/2014]. 33

[3] S. Bowne. Three generations of dos attacks (with audience participation, as victims). http://samsclass.info/defcon.html, 2011. [Acessado em 12/06/2013]. 4

[4] CERT CC. Trends in denial of service attack technology. *http://www.cert.org/archive/pdf/DoStrends.pdf*, 2001. 9

[5] The Hackers Choice. Thc-ssl-dos. https://www.thc.org/thc-ssl-dos/. [Acessado em 12/02/2013]. 30

[6] P. J. Criscuolo. Distributed denial of service trin00, tribe flood network, tribe flood network 2000, and stacheldraht. http://ftp.se.kde.org/pub/security/csir/ciac/ciacdocs/ciac2319.txt, 2000. [Acessado em 31/07/2013]. 10

[7] Goodin D. Tool lets low-end pc crash much more powerful webserver. http://www.theregister.co.uk/2011/10/24/ssl_dos_tool_released/. [Acessado em 10/02/2014]. 30

[8] T. Dierks and C. Allen. The tls protocol version 1.0, 1 1999. 2, 17, 21, 25

[9] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2, 8 2008. 2, 17, 21, 25

[10] C. Douligeris and D.N. Serpanos. *Network Security: Current Status and Future Directions*. Wiley, 2007. 2

[11] The Apache Software Foundation. Welcome to the apache software foundation! http://www.apache.org/. [Acessado em 30/04/2014]. 39

[12] Wireshark Foundation. Wireshark™ *web page*. http://www.wireshark.org/. [Acessado em 30/04/2014]. 39

[13] O Globo. O escândalo da espionagem dos eua. http://oglobo.globo.com/mundo/o-escandalo-da-espionagem-dos-eua-10191175. [Acessado em 08/04/2014]. 30, 56

[14] ha.ckers.org. Slowloris http dos. http://ha.ckers.org/slowloris/, 2009. [Acessado em 15/12/2012]. 4

[15] J.L. Harrington. *Network Security: A Practical Approach*. Morgan Kaufmann Series in Networking. Elsevier, 2005. 2

[16] C. Hock-Chuan. Http, security with ssl. http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_SSL.html, 2009. [Acessado em 11/04/2013]. vii, ix, 26, 27

[17] IBM®. Cipher suite definitions. http://pic.dhe.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.gska100%2Fcsdcwh.htm. [Acessado em 17/04/2013]. ix, 22

[18] J.F. Kurose and K.W. Ross. *Computer Networking: A Top-Down Approach*. Always learning. Pearson, 2012. 1, 3

[19] Brown M. Boost network performance with libevent and libev. http://www.ibm.com/developerworks/aix/library/au-libev/, 2010. [Acessado em 22/04/2014]. 46

[20] Horowitz M. Perfect forward secrecy can block the nsa from secure web pages, but no one uses it. http://blogs.computerworld.com/encryption/22366/can-nsa-see-through-encrypted-web-pages-maybe-so, 2013. [Acessado em 08/04/2014]. 30, 31, 32

[21] J. Mirkovic and P. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. 2004. 3, 6, 7, 8, 9, 12, 13, 14, 15, 16

[22] A. Mitokrotsa and C. Douligeris. Ddos attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, 44(5):643 – 666, 2004. 3, 5, 6, 7, 8, 10, 12, 13

[23] J. Mogul, Compaq, H. Frystyk, W3C/MIT, L. Masinter, Xerox, P. Leach, Microsoft, T. Berners-Lee, W3C/MIT, Compaq/W3C, UC Irvine, J. Gettys, and R. Fielding. Hypertext transfer protocol – http/1.1. RFC 2616, 1999. 4

[24] Mathewson N. and Provos N. Libevent – an event notification library. http://libevent.org/. [Acessado em 22/04/2014]. 33, 34

[25] Cisco®Security Intelligence Operations. A cisco guide to defending against distributed denial of service attacks. http://www.cisco.com/web/about/security/intelligence/guide_ddos_defense.html#_Toc374453049. [Acessado em 31/07/2013]. vii, 4, 10, 11

[26] Microsoft®patterns & practices. Data confidentiality. http://msdn.microsoft.com/en-us/library/ff650720.aspx, 2005. [Acessado em 11/04/2013]. vii, 19

[27] D. Piscitello. Anatomy of a dns ddos amplification attack. http://www.watchguard.com/infocenter/editorial/41649.asp, 2013. [Acessado em 25/08/2013]. 11

[28] Digia plc®. Qt project. http://qt-project.org/. [Acessado em 22/04/2014]. 33, 34

[29] RecordsBackground.com. American tech companies wasting their time with perfect forward secrecy. http://blog.recordsbackground.com/2013/11/23/american-tech-companies-wasting-their-time-with-perfect-forward-secrecy/, 2013. [Acessado em 08/04/2014]. 31

[30] M. Rouse. Asymmetric cryptography (public-key cryptography). http://searchsecurity.techtarget.com/definition/asymmetric-cryptography, 2008. [Acessado em 11/04/2013]. 18

[31] Benutzer S. Diffie-hellman-schlüsselaustausch. http://pt.wikipedia.org/wiki/Ficheiro:Diffie-Hellman-Schl%C3%BCsselaustausch.png, 2006. [Acessado em 5/01/2013]. vii, 28

[32] M. Slaviero. Tls/ssl and .net framework 4.0. https://www.simple-talk.com/dotnet/.net-framework/tlsssl-and-.net-framework-4.0/, 2011. [Acessado em 1/11/2013]. vii, 17, 21

[33] Bernat V. Ssl computational dos mitigation. http://vincent.bernat.im/en/blog/2011-ssl-dos-mitigation.html. [Acessado em 12/02/2013]. vii, 30, 31, 32, 34, 35, 36

[34] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: Global characteristics and prevalence. *In Proceedings of the 2003 ACM SIGMETRICS International conference on Measurement and Modeling of Computer Systems*, pages 138–147, 2003. 9